UNIVERSIDAD DE GRANADA

DESARROLLO Y OPTIMIZACION DE NUEVOS MODELOS DE REDES NEURONALES BASADAS EN FUNCIONES DE BASE RADIAL

TESIS DOCTORAL

Mouncef Filali Bouami

Granada, 2005

Departamento de Arquitectura y Tecnología de Computadores

Editor: Editorial de la Universidad de Granada
Autor: Mouncef Filali Bouami
D.L.: Gr. 994 - 2005
ISBN: 84-338-3419-3

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Índice de acrónimos

ADALINE	**ADA**ptive **LIN**ear **E**lement
adjswapMutation	**adj**acent **swap Mutation**, mutación mediante trueque entre adyacentes
AG	**A**lgoritmos **G**enéticos
AI	**A**rtificial **I**ntelligence
BDA	**B**isector **D**el **Á**rea
CPU	**C**entral **P**rocessing **U**nit
CDA	**C**entro **D**el **Á**rea
CIV	**C**losest **I**nput **V**ector
CFA	**C**lustering for **F**unction **A**pproximation
dec_η	**dec**remento del coeficiente de aprendizaje η
ELBG	**E**nhanced **L**inde-**B**uzo-**G**ray
XOR	E**X**clusive **OR**
FIS	**F**uzzy **I**nference **S**ystem, sistema de inferencia difusa
inc_η	**inc**remento del coeficiente de aprendizaje η
KNN	**k-N**eighbors **N**etworks
LMS	**L**east **M**ean **S**quare
LSE	**L**east **S**quares **E**stimator
MATLAB	**MAT**rix **LAB**oratory
MSE	**M**ean **S**quared **E**rror
MDM	**M**edia **D**el **M**áximo

MF	**M**embership **F**unction, función de pertenencia
MLP	**M**ulti-**L**ayer **Perceptrón**
NRMSE	**N**ormalized **R**oot **M**ean **S**quared **E**rror
RBF	**R**adial **B**asis **F**unction
RBFN	**R**adial **B**asis **F**unction **N**etwork
SVD	**S**ingular **V**alue **D**ecomposition
S-RBF	**S**um **R**adial **B**asis **F**unction
unifMutation	**Unif**orm **Mutation**, mutación uniforme
VLSI	**V**ery **L**arge **S**cale **I**ntegrated Circuits

Capítulo 1/
Introducción al soft-computing

1.1. Introducción

En este trabajo se presenta un método novedoso llamado S-RBF (Sum Radial Basis Function) y su efectividad en problemas de clasificación de muestras, la aproximación de funciones, y la estimación de series temporales.

S-RBF utiliza un algoritmo genético para encontrar, de forma automática una red de funciones de base radial (*Radial Basis Function,* RBF) que resuelva un problema dado, dentro de una arquitectura integrada, y teniendo en cuenta como principal objetivo la minimización del error y el coste computacional al realizar esta tarea.

La construcción de une red de funciones base radial (*Radial basis function network*, RBFN) se basa en la determinación de sus diferentes parámetros como el número de neuronas de su capa oculta, la posición, el ancho y el peso de conexión de cada función que opera dentro de dicha neurona, a continuación, hay que resolver un sistema de ecuaciones para obtener los valores óptimos para los pesos de conexión. Esta es, quizás la mayor diferencia entre las RBFNs y los demás modelos de redes neuronales artificiales, en los que el mayor problema es diseñar un algoritmo que

permita estimar los pesos de conexiones sinápticas. La tarea de diseñar una red RBFN se convierte en un problema de optimización muy apropiado para ser abordado mediante técnicas de computación bio-inspirada.

El algoritmo S-RBF es una combinación de un algoritmo genético que lleva a cabo la búsqueda global por un lado, y de una RBFN, que aprovechando las características de los operadores de T-norma mejora el entrenamiento de ésta y la determinación de los parámetros intrínsecos que la caracterizan.

La efectividad del algoritmo S-RBF ha sido probada aplicándolo a diversos problemas, tanto sintéticos como reales. Los resultados obtenidos han sido comparados con los que ofrecen otras técnicas de clasificación, aproximación de funciones y de estimación de series temporales. Como conclusión general, el algoritmo S-RBF proporciona unos buenos resultados en la mayoría de los experimentos llevados a cabo, con un coste computacional muy bajo.

1.2. Los Algoritmos genéticos

Los algoritmos genéticos [Holland, 1975] [Goldberg, 1994] [Davis, 1991] son algoritmos de búsqueda estocásticos que imitan el proceso de selección natural y la genética. Han sido propuestos para estudiar los procesos adaptativos de los sistemas naturales, y diseñar sistemas de software con comportamiento adaptativo. Los algoritmos genéticos han sido aplicados en varios problemas de optimización con gran éxito [De Jong, 1975] [De Jong, 1985] [Michalewicz, 1999] [Vignaux & Michalewicz, 1991] [Goldberg, 1989b].

1.2.1. Descripción del método

Los algoritmos genéticos son iterativos, conservan un conjunto de soluciones en cada iteración. Inicialmente, el conjunto de soluciones es generado aleatoriamente y en cada iteración, los operadores genéticos forman un nuevo conjunto de soluciones imitando los principios de la evolución y la herencia. Se evalúa cada solución mediante

una función objetivo, y se repite este proceso hasta alcanzar alguna forma de convergencia. El nuevo conjunto de soluciones puede componerse de antiguas soluciones que consigan un alto valor mediante la función objetivo, o nuevas soluciones generadas por la combinación de otras soluciones previas.

El vocabulario usado en los algoritmos genéticos es adaptado de la genética natural. Una solución se le llama *cromosoma*. El cromosoma se compone de genes. La función objetivo se llama *función de evaluación*, y un conjunto de soluciones se llama *población*. Una iteración de un algoritmo genético se llama *generación*.

En su forma más simple, un algoritmo genético requiere una representación en cadena de caracteres para codificar los parámetros, una función de evaluación, un conjunto de operadores genéticos para crear nuevos individuos, y un conjunto de probabilidades para controlar los operadores genéticos.

1.2.2. Componentes y funcionamiento de un algoritmo genético

Los algoritmos genéticos han sido propuestos para resolver problemas lineales y no-lineales explorando todas las áreas mediante la mutación, el cruce y las operaciones de selección aplicadas a los individuos de una población [Michalewicz 1999].

La estructura típica de un algoritmo genético es:

1. inicialización. Generar aleatoriamente una población de soluciones.
2. evaluar cada cadena de soluciones basándose en la función de evaluación.
3. aplicar el conjunto de operadores genéticos para generar una nueva población de soluciones.
4. repetir los pasos 2 y 3 hasta alcanzar una solución convergente.

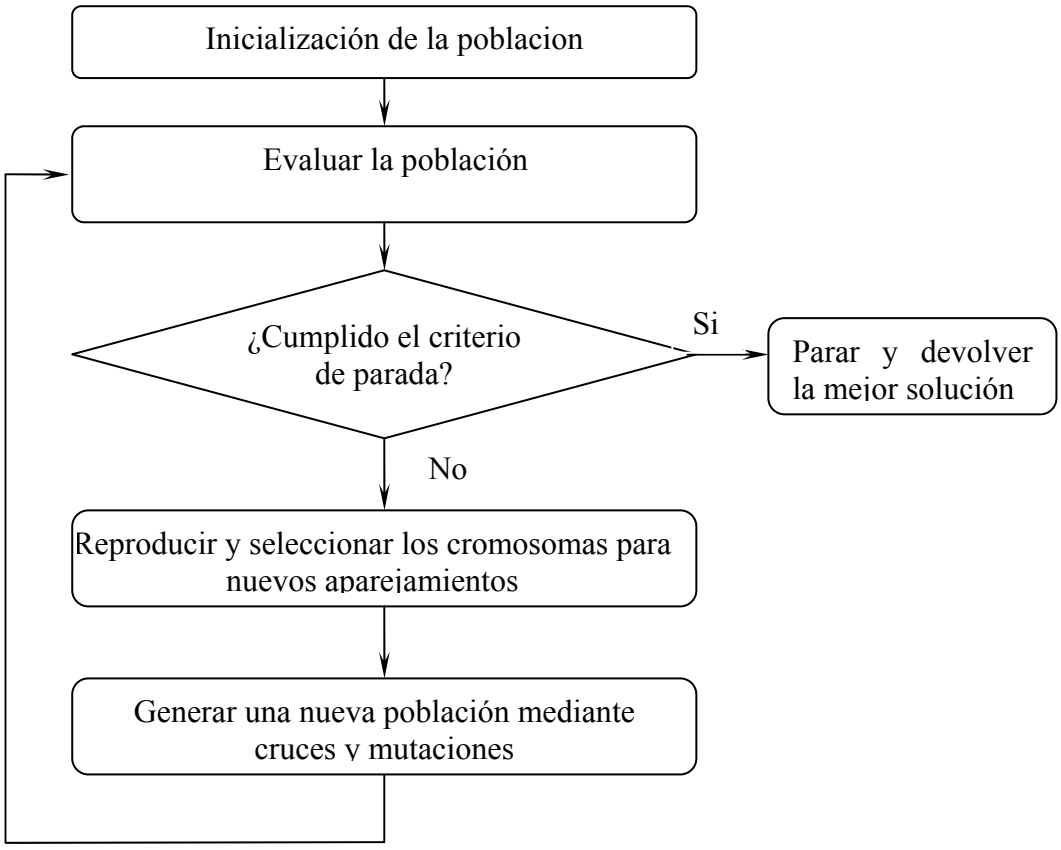
La clave para un algoritmo genético fiable es la buena representación de la cadena de soluciones y los operadores genéticos.

Los algoritmos genéticos, usando la evolución simulada, buscan las mejores soluciones. Generalmente, los mejores individuos de cualquier población tienden a sobrevivir y reproducirse para la siguiente generación, con ello una mejora de las

generaciones sucesivas. Sin embargo, los individuos inferiores pueden, fortuitamente, sobrevivir y reproducirse.

El uso de lo algoritmos genéticos requiere la determinación de seis parámetros fundamentales, la representación en cromosomas, la función de selección, los operadores genéticos que completan la función de reproducción, la creación de una población inicial, el criterio de parada, y la función de evaluación. A continuación una descripción de cada uno de estos parámetros.

El algoritmo 1.1 muestra una descripción detallada de los pasos básicos de un algoritmo genético.



Algoritmo 1.1: *Las etapas básicas de un algoritmo genético.*

1.2.2.1. Representación de la solución

Usando la noción de selección natural, el operador de reproducción genera una nueva población desde individuos de la antigua población. La selección no es totalmente aleatoria sino que se basa en la aptitud relativa de un individuo respeto a la aptitud de la población global.

Suponer una población con seis individuos, y cada uno con su aptitud:

Individuo (i)	Valor de la función de evaluación (F_i)	$F_i / \sum_j F_j$
A	25	.5
B	10	.2
C	6	.12
C	6	.12
D	2	.04
E	1	.02
Total	50	1.

Tabla 1.1: *Ejemplo de una distribución en la que cada individuo tiene una aptitud propia.*

Notar que los individuos en una población no tienen que ser únicas. La proporción $F_i / \sum_j F_j$ representa la probabilidad de que el i-ésimo individuo sobreviva a la próxima generación. Esto significa que los individuos con mayor aptitud tendrían mayor probabilidad de supervivencia, además se mantiene constante el número de individuos en una población a lo largo de la ejecución del algoritmo, y el operador de reproducción genera poblaciones del mismo tamaño, implicando, finalmente, que los individuos con mayor aptitud dominan la población.

Para cualquier algoritmo genético, se necesita una representación en cromosoma para describir cada individuo de una población de interés. El esquema de representación

determina la estructura de un problema en los algoritmos genéticos y determina también los operadores genéticos que se van a usar. Cada individuo o cromosoma es constituido de genes de algún alfabeto. Una alfabeto puede consistir en dígitos binarios (0 y 1), enteros, símbolos (i.e., A, B, C, D), matrices, etc. En el diseño original de [Holland 1975] el alfabeto estaba limitado a los dígitos binarios. Desde entonces, la representación de problemas ha sido tema de muchas investigaciones. Se ha demostrado que las representaciones más naturales son más eficaces y producen mejores soluciones [Michalewicz 1999]. Hay experimentaciones extensivas que comparan los algoritmos genéticos binarios de los de valores reales y demuestran que los algoritmos genéticos con valores reales son más eficientes en términos de tiempo en la CPU. Se ha demostrado igualmente que con valores reales, un problema es mejor representado lo que permite ganar en precisión con resultados consistentes a lo largo de las replicaciones.

1.2.2.2. La función de selección

La selección de individuos para producir generaciones sucesivas tiene un papel extremadamente importante en un algoritmo genético. Se realiza una selección probabilística basada en la aptitud de los individuos de manera que los mejores individuos tienen más posibilidad de ser escogidos. Un individuo de una población puede ser seleccionado más de una vez entre la totalidad de individuos con posibilidad de ser escogidos para producir la próxima generación. Existen varios esquemas para el proceso de selección: la selección por ruleta y sus extensiones, las técnicas de ajuste, el torneo, los modelos elitistas, y los métodos de clasificación [Goldberg 1989, Michalewicz 1999].

Un enfoque de selección común asigna una probabilidad de selección P_j, a cada individuo j basándose en su estimación de aptitud. Se generan N series de números aleatorios y se comparan respeto a la probabilidad global $C_i = \sum_{j=1}^{i} P_j$, de la población. Se selecciona el individuo apropiado i y se añade a la nueva población si

$C_{i-1} < U(0,1) \leq C_i$. existen varios métodos de asignar probabilidades a los individuos: la ruleta, la clasificación lineal y la clasificación geométrica.

La ruleta, desarrollada por [Holland 1975] fue el primer método de selección. La probabilidad P_j de cada individuo se define como:

$$P[\text{se escoja el individuo } i] = \frac{F_i}{\sum_{j=1}^{TamPob} F_j}, \tag{1.1}$$

Donde F_i es la estimación de aptitud del individuo i. El uso de la selección por ruleta limita el algoritmo genético a la tarea de maximización puesto que la función de evaluación ha de organizar las soluciones en un conjunto de $\Re^+$ completamente ordenado.

Los métodos de clasificación requieren la función de evaluación solamente para organizar las soluciones en un conjunto parcialmente ordenado, permitiendo trabajar en minimización y negatividad.

Los métodos de clasificación asignan la probabilidad P_i basándose en el rango de la solución i cuando se ordena todas las soluciones. La clasificación geométrica normalizada [Joines & Houck 1994], define P_i para cada individuo como:

$$P[\text{se escoja el } i-\text{ésimo individuo}] = q'(1-q)^{r-1} \tag{1.2}$$

Donde:

q es la probabilidad de seleccionar el mejor individuo,

r es el rango del individuo, con 1 como mejor rango,

P es el tamaño de la población,

$$q' = \frac{q}{1-(1-q)^P}.$$

La selección por torneo, igual que los métodos de clasificación, requiere solamente la función evaluación para organizar la soluciones en un conjunto parcialmente ordenado, sin embrago no asigna probabilidades. La selección por torneo, funciona escogiendo *j* individuos aleatoriamente, con reemplazo, de la población, e inserta los mejores en la nueva población. Se repite este proceso hasta la selección de *N* individuos.

1.2.2.3. Los operadores genéticos

Los operadores genéticos proporcionan el mecanismo básico de búsqueda de los algoritmos genéticos. Se usan los operadores para crear nuevas soluciones basándose en las ya existentes en una población. Existen dos tipos básicos de operadores: el cruce y la mutación. El cruce toma dos individuos y devuelve dos nuevos individuos, mientras la mutación altera un individuo para producir una nueva solución. La aplicación de estos tipos de operadores y sus derivados depende de la representación del cromosoma utilizado.

Supongamos $\overline{X}$ y $\overline{Y}$, dos vectores *m*-dimensionales, que representan individuos (padres) de la población. Para $\overline{X}$ y $\overline{Y}$ binarios, definimos los siguientes operadores: mutación binaria y cruce simple.

El operador de cruce utiliza el concepto de acoplamiento donde se combinan los genes de diferentes individuos para producir un nuevo individuo. Puesto que los genes representan las características de un individuo, si se combinan las "buenas" características de diferentes individuos, el individuo generado tendría incluso mejores características.

Asumiendo que los genes de un individuo están ordenados en una línea recta, entonces el cruce de dos individuos se puede realizar mediante los siguientes pasos señalados en la figura 1.1.

1. seleccionar aleatoriamente un punto (punto de cruce) para romper el individuo en dos partes.
2. intercambiar las segundas partes de los individuos.

Se obtiene por lo tanto dos nuevos individuos con materiales genéticos de sus padres. Este operador de cruce es un método potente para intercambiar información y crear nuevas soluciones. Notar que si la población tiene un único tipo de individuos, el operador de cruce no puede generar nuevos individuos. Para remediar a esta situación, se introduce el operador de mutación.

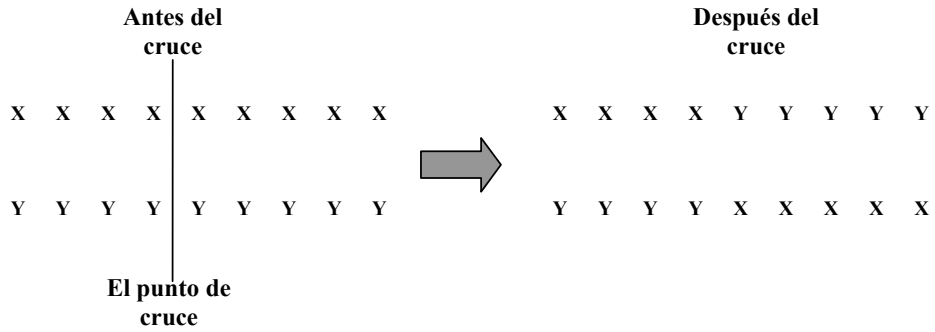


Figura 1.1: *Ejemplo de un cruce simple entre dos cromosomas.*

El operador de mutación selecciona aleatoriamente un gene de un individuo, y altera su valor creando un nuevo individuo, figura 1.2 el operador de mutación introduce una variabilidad dentro de la población, y proporciona un mecanismo para escapar de los mínimos locales. Notar que un algoritmo que usa únicamente el operador de mutación sería equivalente a un algoritmo de búsqueda aleatoria.

La mutación binaria permuta cada gene de cada individuo de una población con probabilidad p_m según la ecuación siguiente:

$$x_i^{'} = \begin{cases} 1 - x_i, \text{ si } U(0,1) < p_m \\ x_i, \text{ sino} \end{cases} \tag{1.3}$$

El cruce simple genera un número aleatorio *r* de una distribución uniforme entre 1 y *m*, y crea dos nuevos individuos ($\overline{X}'$ y $\overline{Y}'$) según las ecuaciones siguientes:

$$x_i^{'} = \begin{cases} x_i, \text{ si } i < r \\ y_i, \text{ sino} \end{cases} \tag{1.4}$$

$$y_i^{'} = \begin{cases} y_i, \text{ si } i < r \\ x_i, \text{ sino} \end{cases} \tag{1.5}$$

Los operadores para representaciones con valores reales, i.e. un alfabetos de flotantes, has sido desarrollados por [Michalewicz 1999]. Para $\overline{X}$ y $\overline{Y}$ reales, se definen los siguientes operadores: la mutación uniforme, la mutación no-uniforme, la mutación multi-no-uniforme, la mutación al límite, el cruce simple, el cruce aritmético, y el cruce heurístico. Sean a_i y b_i el límite inferior y superior, respectivamente, de cada variable i.

La mutación uniforme selecciona aleatoriamente una variable *j*, y la sustituye por un valor aleatorio uniforme $U(a_i, b_i)$:

$$x_i^{'} = \begin{cases} U(a_i, b_i) \text{ si } i = j \\ x_i \text{ sino} \end{cases} \tag{1.6}$$

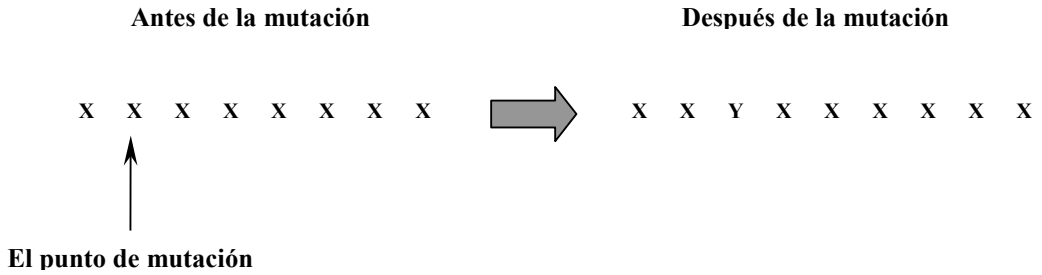


Figura 1.2: *Ejemplo de una mutación aplicada a un cromosoma.*

La mutación al límite selecciona aleatoriamente una variable *j* y la sustituye por el valor del límite inferior o superior, de la manera siguiente:

$$x_i^{'} = \begin{cases} a_i, \text{ si } i = j, r < 0.5 \\ b_i, \text{ si } i = j, r \geq 0.5 \\ x_i, \text{ sino} \end{cases} \tag{1.7}$$

Donde $r = U(0,1)$.

La mutación no-uniforme selecciona aleatoriamente una variable *j* y la sustituye por un número aleatorio no-uniforme:

$$x_i^{'} = \begin{cases} x_i + (b_i - x_i)\, f(G), \text{ si } r_1 < 0.5, \\ x_i - (x_i + a_i)\, f(G), \text{ si } r_1 \geq 0.5, \\ x_i, \text{ sino} \end{cases} \tag{1.8}$$

Donde

$$f(G) = \left(r_2 \left(1 - \frac{G}{G_{\max}} \right) \right)^b ,$$

r_1 y r_2 son números aleatorios uniformes comprendidos entre (0,1),

G es la generación actual y $G_{\max}$ es el número máximo de generaciones,

b es el parámetro de ancho.

El operador de mutación multi-no-uniforme, aplica el operador no-uniforme a todas las variables del padre $\overline{X}$.

El cruce simple con valores reales es idéntico a la versión binaria presentada anteriormente en las ecuaciones (1.4) y (1.5) .El cruce aritmético produce dos combinaciones lineales añadidas:

$$\overline{X}' = r\overline{X} + (1-r)\overline{Y} \tag{1.9}$$

$$\overline{Y}' = (1-r)\overline{X} + r\overline{Y} \tag{1.10}$$

Donde $r = U(0,1)$.

El cruce heurístico produce una extrapolación lineal de dos individuos. Es el único operador que utiliza información sobre la aptitud de los individuos. Se crea un nuevo individuo $\overline{X}'$ usando la ecuación (1.11) donde $r = U(0,1)$ y $\overline{X}$ es mejor que $\overline{Y}$ en términos de aptitud. Si $\overline{X}'$ no es factible, es decir, la *factibilidad* es igual a 0 como muestra la ecuación (1.13), entonces el algoritmo genera un nuevo número ***r*** aleatorio y crea una nueva solución usando la ecuación (1.11), o sino se para.

$$\overline{X}' = \overline{X} + r\left(\overline{X} - \overline{Y}\right) \tag{1.11}$$

$$\overline{Y}' = \overline{X} \tag{1.12}$$

$$factibilidad = \begin{cases} 1, \text{ si } x_i' \geq a_i, x_i' \leq b_i \ \forall i \\ 0, \text{ sino} \end{cases} \tag{1.13}$$

1.2.2.4. La inicialización, terminación y las funciones de evaluación.

El primer paso en un algoritmo genético es la constitución de una población inicial, el método más común es generar aleatoriamente soluciones para la población total. Sin embargo, desde que los algoritmos genéticos pueden mejorar continuamente las soluciones existentes (es decir, las soluciones desde otras heurísticas y/o prácticas), se puede dotar la población inicial de soluciones potencialmente buenas, el resto de soluciones siendo generado aleatoriamente.

El algoritmo genético funciona, de generación en generación, seleccionando y reproduciendo padres hasta coincidir con la condición de parada. Una condición utilizada más frecuentemente es especificar un número máximo de generaciones. Otra

estrategia de terminación implica la convergencia de la población. Generalmente, los algoritmos genéticos fuerzan la convergencia de la totalidad de la población. Cuando la suma de desviaciones de los individuos es inferior a un umbral especificado, se puede detener el algoritmo. La falta de progresión de la mejor solución a lo largo de un número de generaciones puede, igualmente, ser un criterio de parada del algoritmo. Otra alternativa consiste en fijar un valor objetivo cuya evaluación se establecería basándose en un umbral "aceptable". Según la naturaleza del problema, se puede adaptar el criterio de parada.

En los algoritmos genéticos, se puede utilizar varias formas de funciones de evaluación, sujetas un mínimo requerimiento, organizar la población en un conjunto parcialmente ordenado. La función de evaluación es independiente del algoritmo genético, es decir, las reglas de decisión estocásticas.

1.2.3. Ejemplo ilustrativo

Considerar el problema de maximización de la función

$$f(x) = 0.4 + \operatorname{sinc}(4x) + 1.1\operatorname{sinc}(4x+2) + 0.8\operatorname{sinc}(6x-2) + 0.7\operatorname{sinc}(6x-4), \quad x \in [-2,2] \tag{1.14}$$

Donde

$$\operatorname{sinc}(x) = \begin{cases} 1 & \text{si } x = 0, \\ \dfrac{\sin(\pi x)}{\pi x} & \text{si } x \neq 0. \end{cases} \tag{1.15}$$

La figura 1.3 muestra un grafico de $f(x)$ dentro del intervalo $[-2,2]$. El máximo de *f* es alcanzado en el punto $x = -0.507179$ con un valor de 1.501564.

Usando un algoritmo genético para maximizar *f*, podemos codificar el rango de x en un número binario representado por un vector binario. Un número binario de 16 bits puede proporcionar una resolución de $\left(2-(-2)\right)/\left(2^{16}-1\right)=0.000061$ por bit.

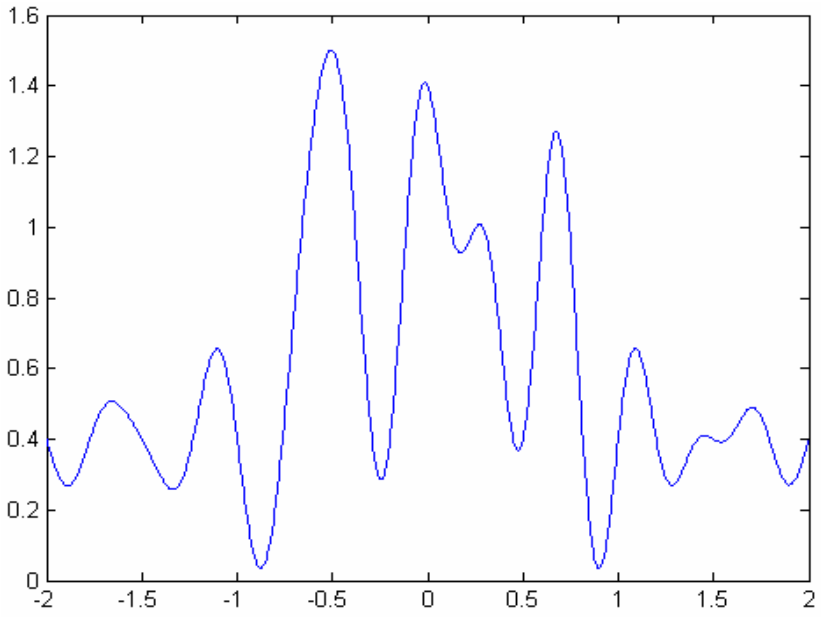


Figura 1.3: *Grafico de una función* $f(x)$ *con varios extremos locales.*

La ecuación (1.16) convierte los elementos del intervalo $[-2,2]$ en números binarios en el intervalo $[0,65535]$.

$$x=-2+\frac{4b}{65535}, \tag{1.16}$$

Donde *b* es un número binario dentro de $[0,65535]$. Por ejemplo,

$$v_1=(1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0)\Rightarrow x=2.93506$$

Y $$v_2=(0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0)\Rightarrow x=-1.87610.$$

Para implementar el algoritmo genético que maximiza la función definida por la ecuación (1.14), se usaran tres operadores genéticos, a saber, la reproducción, el cruce, y la mutación.

Se implementa el operador de reproducción mediante el método de ruleta tal como se describe anteriormente. El operador de cruce junta dos vectores binarios, y crea dos nuevos vectores binarios. Se utilizara la forma de cruce simple donde se selecciona un punto aleatorio entre dos bits sucesivos (punto de cruce). Se forman los dos vectores binarios se forman intercambiando las segundas partes de ambos vectores.
Por ejemplo, si se selecciona el siguiente punto de cruce (señalado por ||):

$$v'_1 = (10111011111||11110)$$
$$v_2 = (00110011111||11110)$$

Entonces los nuevos vectores binarios serán:

$$v'_1 = (10111011111||11110)$$
$$v_2 = (00110011111||00110)$$

La mutación es muy sencilla. En un vector binario dado, se selecciona aleatoriamente un bit y se invierte su valor. Por ejemplo, se selecciona el bit subrayado en el vector v_1, entonces el nuevo vector sería:

$$v_1 = (1011101111\underline{1}00110)$$
$$v'_1 = (10111011111000110)$$

Adoptando los siguientes parámetros:

- tamaño de población = 30,

- probabilidad de cruce = 0.3,
- probabilidad de mutación = 0.01.

En 100 generaciones y adoptando los parámetros adecuados para los operadores genéticos, se alcanza como resultado un máximo global considerado optimo.

1.3. Los Conjuntos Difusos

En este apartado se introducen las definiciones básicas, las anotaciones y las operaciones dentro de los conjuntos difusos.

1.3.1. Introducción

Los conjuntos clásicos han demostrado ser una importante herramienta en las matemáticas y las ciencias de la computación, sin embargo, no reflejan lo que son los pensamientos y los conceptos de la naturaleza humana que suelen ser abstractos e imprecisos. A diferencia del conjunto clásico, un conjunto difuso, como indica su nombre, es un conjunto sin frontera concisa, la transición de "perteneciente al conjunto" a "no perteneciente al conjunto" es gradual, y esta transición suave se ilustra con funciones de pertenencia que dan una flexibilidad a los conjuntos difusos para el modelado de expresiones lingüísticas, como viene señalado en el trabajo de Zadeh [Zadeh 1965] "los conjuntos difusos", estos conjuntos definidos de manera imprecisa " juegan un papel determinante en la manera de pensar de los humanos, particularmente en el reconocimiento de muestras, la comunicación de información y la abstracción "

1.3.2. Definiciones y terminologías básicas

Definición 1.1. Conjuntos difusos y funciones de pertenencia

Si X es un grupo de elementos generalmente llamados x, pues el conjunto difuso A en X esta definido como un conjunto de parejas ordenadas:

$$A = \left\{ \left(x, \mu_A \left(x \right) \right) \middle| x \in X \right\} \tag{1.17}$$

$\mu_A \left(x \right)$ es la función de pertenencia del conjunto difuso A, la función de pertenencia asigna a cada elemento de X un grado de pertenencia contenido entre 0 y 1.
Esta definición de un conjunto difuso es una simple extensión de la definición de un conjunto clásico con la posibilidad para la función característica de tener cualquier valor entre 0 y 1.

La construcción de un universo difuso depende de dos factores, la definición de un universo de discurso conveniente y la especificación de funciones de pertenencia apropiadas, este ultimo factor es *subjetivo*, las funciones de pertenencia especificadas para un mismo concepto pueden variar considerablemente según cada persona, esta subjetividad viene del carácter individual del percebimiento y de la expresión de los conceptos abstractos y sin embrago tiene poco que ver con la aleatoriedad, por lo tanto, la subjetividad y la no-aleatoriedad de los conjuntos difusos es la principal diferencia entre el estudio de los conjuntos difusos y la teoría de la probabilidad.

1.3.3. Operaciones teóricas

La unión, la intersección y el complemento son las operaciones básicas en los conjuntos clásicos, a partir de estas operaciones se puede establecer algunas identidades.

Definición 1.2. Producto y coproducto Cartesiano.
A y B son dos conjuntos difusos, el producto Cartesiano de A y B denotado $A \times B$ es un conjunto difuso con la función de pertenencia:

$$\mu_{A \times B}(x, y) = \min(\mu_A(x), \mu_B(x)) \tag{1.18}$$

Asimismo, el coproducto Cartesiano $A + B$ es un conjunto difuso con una función de pertenencia:

$$\mu_{A\times B}(x,y) = \min(\mu_A(x), \mu_B(x)) \tag{1.19}$$

$A\times B$ y $A+B$ se caracterizan por sus funciones de pertenencia bidimensionales.

Ley de contradicción	$A \cap \overline{A} = \varnothing$
Ley del medio excluido	$A \cup \overline{A} = X$
Identidad	$A \cap A = A,\ A \cup A = A$
Involución	$\overline{\overline{A}} = A$
conmutatividad	$A \cap B = B \cap A,\ A \cup B = B \cup A$
Asociabilidad	$(A \cup B) \cup C = A \cup (B \cup C)$ $(A \cap B) \cap C = A \cap (B \cap C)$
Distributividad	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
Absorción	$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$
Absorción del complemento	$A \cup (\overline{A} \cap B) = A \cup B$ $A \cap (\overline{A} \cup B) = A \cap B$
Leyes de DeMorgan	$\overline{A \cup B} = \overline{A} \cap \overline{B},\ \overline{A \cap B} = \overline{A} \cup \overline{B}$

Tabla 1.2: *Identidades básicas en los conjuntos clásicos, A,B y C son conjuntos clásicos, $\overline{A},\overline{B}$, y $\overline{C}$ son sus complementos correspondientes; X es el universo; y $\varnothing$ es el conjunto vacío.*

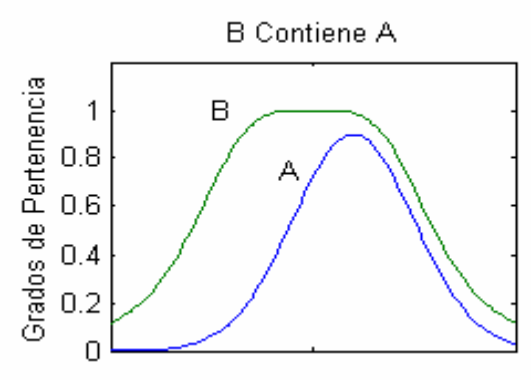


Figura 1.4: El *concepto de* $A \subseteq B$

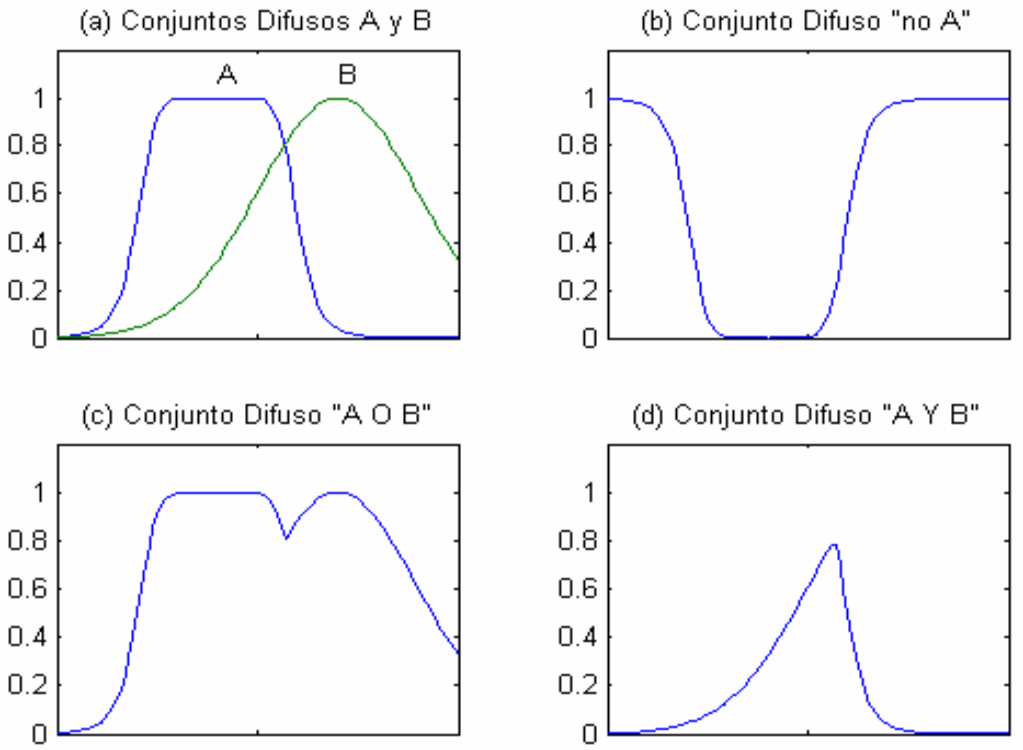


Figura 1.5: *Operaciones sobre conjuntos difusos: (a) dos conjuntos difusos* A *y* B*; (b)* $\overline{A}$*, el complemento de* A*; (c)* $A \cup B$*, la unión de* A *y* B*; (d)* $A \cap B$*, la intersección entre* A *y* B *.*

1.3.4. Reglas difusas y razonamiento difuso

En este párrafo se introducen conceptos del principio de extensión y las relaciones difusas, que expanden las nociones de aplicación de los conjuntos difusos mencionados previamente. A continuación se presenta una definición de las variables lingüísticas y los valores lingüísticos y se explica su uso en las reglas difusas, que son una herramienta de gran utilidad en el modelado cuantitativo de las palabras o sentencias del lenguaje artificial. Interpretando las reglas difusas como relaciones difusas apropiadas, se investigan diferentes esquemas del razonamiento difuso, donde los procesos de inferencia basados en el concepto de regla de inferencia se usan para sacar conclusiones de un conjunto de reglas difusas.

1.3.4.1. El principio de extensión y las relaciones difusas

Las reglas difusas y el razonamiento difuso son la base de los sistemas de inferencia difusa, que a su vez son la herramienta de modelado más importante basada en la teoría de los conjuntos difusos. Han sido aplicados con éxito en muchos campos como el control automático, los sistemas expertos, el reconocimiento de muestras, la predicción de series temporales y la clasificación de datos.

1.3.4.1.1. El principio de extensión

El principio de extensión [Zadeh 1965] y [Zadeh 1975] es un concepto básico de la teoría de los conjuntos difusos que nos permite extenderlas expresiones matemáticas ya aplicadas en los entornos clásicos a los entornos difusos, este proceso generaliza la correspondencia entre puntos de una función $f(.)$ a los conjuntos difusos; supongamos que f es una función de X a Y y A es un conjunto difuso en X definido por:

$$A = \mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \cdots + \mu_A(x_n)/x_n$$

El principio de extensión afirma que la imagen del conjunto difuso A según la proyección de $f(.)$ es un conjunto difuso B:

$$B = f(A) = \mu_A(x_1)/y_1 + \mu_A(x_2)/y_2 + \cdots + \mu_A(x_n)/y_n$$

Donde $y_i = f(x_i)$, $i = 1,\ldots,n$ es decir, el conjunto difuso B puede ser definido a través de los valores de $f(.)$ en $x_1,\ldots,x_n$.

Ejemplo 1.1. *Aplicaciones del principio de extracción a los conjuntos difusos con valores discretos.*

Sea $$A = 0.1/-2 + 0.4/-1 + 0.8/0 + 0.9/1 + 0.3/2$$

Y $$f(x) = x^2 - 3$$

Aplicándole principio de extensión:

$$\begin{aligned} B &= 0.1/1 + 0.4/-2 + 0.8/-3 + 0.9/-2 + 0.3/1 \\ &= 0.8/-3 + (0.4 \vee 0.9)/-2 + (0.1 \vee 0.3)/1 \\ &= 0.8/-3 + 0.9/-2 + 0.3/1, \end{aligned}$$

Donde $\vee$ representa el máximo, la figura 1.6 ilustra este ejemplo.

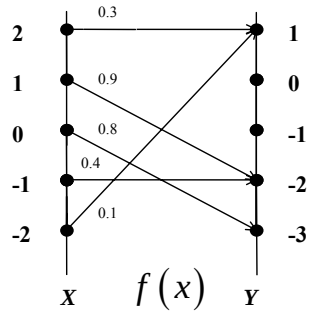


Figura 1.6: *El principio de extensión en los conjuntos difusos con universo discontinuo.*

Se aplica el mismo procedimiento para todo conjunto difuso con universo de discurso continuo X

1.3.4.1.2. Las relaciones difusas

Las relaciones difusas binarias [Zadeh 1965] y [Zadeh 1971-bis] son conjuntos difusos dentro de $X \times Y$ que asignan a cada elemento de $X \times Y$ un grado de pertenencia entre 0 y 1, del mismo modo, las relaciones difusas unitarias, son conjuntos difusos unidimensionales, las aplicaciones de las relaciones difusas incluyen áreas como el control difuso y la toma de decisiones.

Definición 1.3. *Relación binaria difusa*

Sea X y Y 2 universos de discurso, entonces

$$rel = \left\{ \left((x,y), \mu_R (x,y) \right) | (x,y) \in X \times Y \right\} \tag{1.20}$$

Es una **relación difusa binaria** dentro de $X \times Y$.

Ejemplo 1.2. *Relaciones difusas binarias*

Sean $X = Y = R^+$ (grupo de elementos reales positivos) y rel = "*y* es muy superior a *x*" la función de pertenencia de rel puede, subjetivamente, ser definida como:

$$\mu_R(x,y) = \begin{cases} \dfrac{y-x}{x+y+2}, & \text{si } y > x. \\ 0, & \text{si } y \le x. \end{cases} \tag{1.21}$$

Si $X = \{3,4,5\}$ y $Y = \{3,4,5,6,7\}$, entonces es más conveniente expresar la relación difusa rel como una matriz relación:

$$rel = \begin{bmatrix} 0 & 0.111 & 0.200 & 0.273 & 0.333 \\ 0 & 0 & 0.091 & 0.167 & 0.231 \\ 0 & 0 & 0 & 0.077 & 0.143 \end{bmatrix}$$

Donde el elemento de la línea i y la columna j es igual al grado de pertenencia entre el *i*-ésimo elemento de X y el *j*-ésimo elemento de Y.

1.3.4.2. Las variables lingüísticas y las reglas difusas "Si - Entonces"

Como apunta [Zadeh 1973], las técnicas convencionales para el análisis de sistemas son ineptas para tratar los sistemas humanísticos cuyos comportamientos están sugestionados por el criterio humano, la percepción y las emociones. Es una manifestación de lo que podríamos llamar el principio de incompatibilidad: "el aumento de complejidad de un sistema reduce nuestra habilidad para realizar afirmaciones precisas y por lo tanto significativas, hasta un punto que la precisión y el significado casi se convierten en características mutualmente exclusivas" [Zadeh 1973].a consecuencia de esta convicción, Zadeh propuso el concepto de las variables lingüísticas [Zadeh 1971] como enfoque alternativo para modelar el pensamiento humano.

A continuación presentamos la definición formal de una variable lingüística y un ejemplo para clarificar esta definición.

Definición 1.4. *Las variables lingüísticas y los valores lingüísticos.*

> Una variable lingüística se caracteriza por un quinteto $(x, T(x), X, G, S)$ donde x es el nombre de la variable, $T(x)$ es el conjunto de términos de x – es el conjunto de los valores lingüísticos de x, X es el universo de discurso, G es una regla sintáctica que genera los términos de $T(x)$, S es una regla semántica que asocia a cada valor lingüístico A su significado $S(A)$, $S(A)$ es un conjunto difuso en X.

El ejemplo siguiente ayuda a clarificar la definición precedente

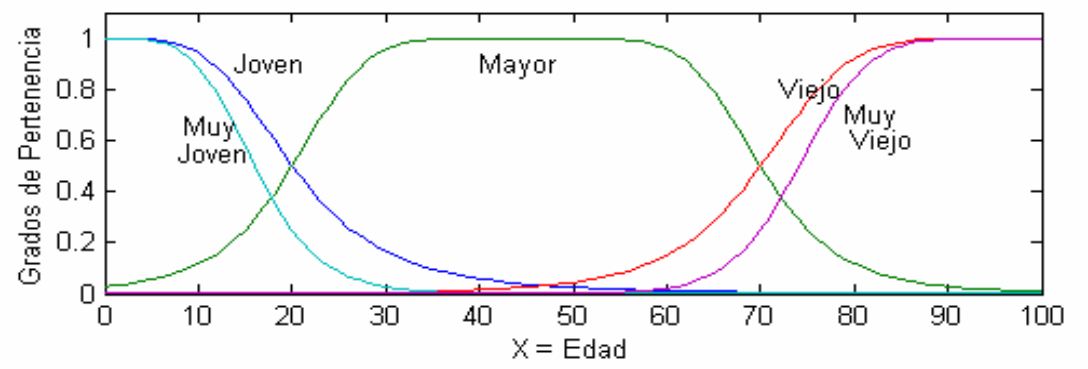


Figura 1.7: *Funciones de pertenencias típicas del conjunto* $T(edad)$

Ejemplo 1.3. *Variables lingüísticas y valores lingüísticos.*

Si interpretamos la *edad* como variable lingüística, entonces el conjunto de términos $T(edad)$ podría ser:

T(edad) = { joven, no joven, muy joven, no muy joven,...,

Mayor, no mayor,...,

Viejo, muy viejo, mas o menos viejo, no muy viejo,..,

No muy viejo y no muy joven,...},

Donde cada término en $T(edad)$ esta caracterizado por un conjunto difuso de un universo de discurso $X = [0,100]$ como muestra la figura 1.7., generalmente, decimos "*edad es joven*" para asignar el valor lingüístico "*joven*" a la variable lingüística "*edad*", en cambio cuando la *edad* esta interpretada como variable numérica, utilizamos la expresión "*edad = 20*" en vez de asignar el valor numérico "*20*" a la variable numérica "*edad*". Las reglas sintácticas se refieren al criterio para generar los valores lingüísticos dentro del conjunto de términos, las reglas semánticas definen la función de pertenencia de cada valor lingüístico, la figura 1.7. representa las funciones de pertenencia más representativas.

Del ejemplo precedente podemos ver que el conjunto de términos consiste en varios términos primarios *(joven, mayor, viejo)* modificados por la negación *("no")* y/o por extensiones *(mas o menos, muy, extremadamente,...)* y por lo tanto vinculados por conjunciones como *y, o, cualquier* y *ningún*, por lo que deberíamos tratar las conjunciones, las negaciones y las extensiones como operadores que cambian el sentido de sus objetos.

Definición 1.5. *Concentración y dilatación de los valores lingüísticos.*

Sea A un valor lingüístico caracterizado por un conjunto difuso con una función de pertenencia $\mu_A(.)$, entonces A^k esta considerado como una versión modificada del valor lingüístico original y se expresa:

$$A^k = \int_X \left[\mu_A(x)\right]^k / x \tag{1.22}$$

Particularmente, la operación de la concentración se define como:

$$CON(A) = A^2 \tag{1.23}$$

Mientras la dilatación se expresa de la forma:

$$DIL(A) = A^{0.5} \tag{1.24}$$

Convencionalmente, tomaremos $CON(A)$ y $DIL(A)$ como resultado de la aplicación de la extensión *muy* y *mas o menos* respectivamente, al término lingüístico A, no obstante hay mas definiciones para las extensiones lingüísticas y con muchas aplicaciones, a continuación definimos el operador de negación **NO** y las conjunciones Y y O como:

$$\text{NO}(A) = \neg A = \int_X \left[1 - \mu_A(x)\right] / x,$$

$$A \text{ Y } B = A \cap B = \int_X \left[\mu_A(x) \wedge \mu_B(x)\right] / x \tag{1.25}$$

$$A \text{ O } B = A \cup B = \int_X \left[\mu_A(x) \vee \mu_B(x)\right] / x$$

Donde A y B son dos valores lingüísticas cuyos significados están definidos respectivamente por $\mu_A(x)$ y $\mu_B(x)$.

Asimilando los operadores precedentes podemos ahora explicar el sentido de los términos lingüísticos compuestos como "*no muy joven y no muy mayor*" y" *joven pero no muy joven*".

Ejemplo 1.4. *Construcción de funciones de pertenencia para los términos lingüísticos compuestos.*

Sean las funciones de pertenencia siguientes que definen el sentido de los términos lingüísticos *joven* y *viejo.*

$$\mu_{joven}(x) = bell(x, 20, 2, 0) = \frac{1}{1 + \left(\frac{x}{20}\right)^4} \tag{1.26}$$

$$\mu_{viejo}(x) = bell(x, 30, 3, 100) = \frac{1}{1 + \left(\frac{x - 100}{30}\right)^6} \tag{1.27}$$

Donde x es la edad de una persona dentro del intervalo $[0,100]$, podemos construir funciones de pertenencia para los siguientes términos lingüísticos compuestos.

- $"mas\ o\ menos\ viejo" = DIL(viejo) = viejo^{0.5}$

$$= \int_X \sqrt{\frac{1}{1+\left(\frac{x-100}{30}\right)^6}} \Big/ x \cdot$$

- $"no\ es\ joven\ y\ no\ es\ viejo" = \neg joven \cap \neg viejo$

$$= \int_X \left[1 - \frac{1}{1+\left(\frac{x}{20}\right)^4}\right] \wedge \left[1 - \frac{1}{1+\left(\frac{x-100}{30}\right)^6}\right] \Big/ x \cdot$$

- $" joven\ pero\ no\ muy\ joven" = joven \cap \neg joven^2$

$$= \int_X \left[\frac{1}{1+\left(\frac{x}{20}\right)^4}\right] \wedge \left[1 - (\frac{1}{1+\left(\frac{x}{20}\right)^4})^2\right] \Big/ x$$

- $"extremadamente\ viejo" = CON\left(CON\left(CON\left(viejo\right)\right)\right) = \left(\left(\left(viejo\right)^2\right)^2\right)^2$

$$= \int_X \left[\frac{1}{1+\left(\frac{x-100}{30}\right)^6}\right]^8 \Big/ x \cdot$$

Extremadamente significa *muy muy muy*. La figura 1.8.(a) muestra las funciones de pertenencia de los términos lingüísticos primarios *joven* y *viejo* y la figura 1.8.(b) muestra las funciones de pertenencia de los términos lingüísticos compuestos.

Una regla difusa de formato "Si-Entonces" también llamadas reglas difusas o implicaciones difusas tiene el formato:

$$\text{Si } x \text{ es } A \text{ entonces } y \text{ es } B \qquad (1.28)$$

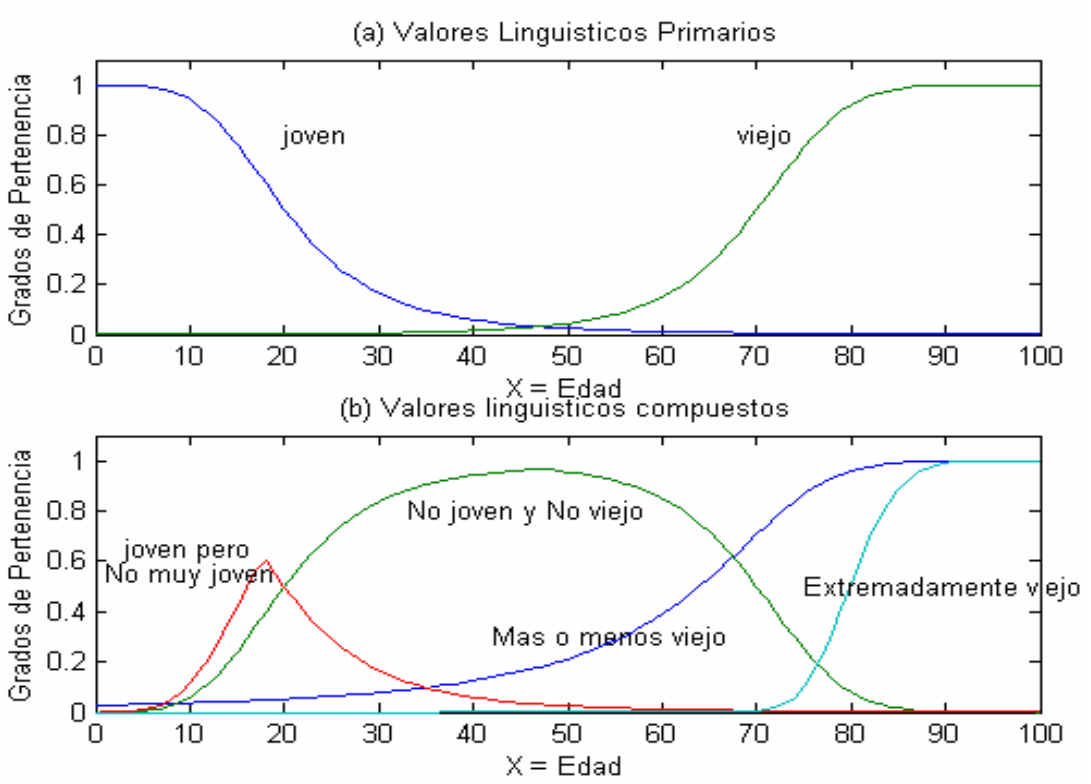


Figura 1.8: *Funciones de pertenencia de valores lingüísticos primarios y compuestos del ejemplo 1.4.*

Donde A y B son valores lingüísticos definidos por conjuntos difusos dentro de universos de discurso X y Y respectivamente. Llamamos "antecedente" la parte "x es A" y "y es B"es la "consecuencia" o la conclusión. Esta expresión se abrevia al formato $A \to B$ y se puede escribir:

$$A \to B = \neg A \cup B = \int_{X \times Y} \left(1 - \mu_A(x)\right) \vee \mu_B(x)$$

1.3.4.3. El razonamiento difuso

El razonamiento difuso, también conocido como razonamiento aproximativo, es un procedimiento de inferencia que saca conclusiones a partir de un conjunto de reglas "Si-Entonces"

Definición 1.6. *Razonamiento aproximativo*

Sean A, A' y B tres conjuntos difusos de X, X y Y respectivamente. Asumimos la implicación difusa $A \to B$ se expresa como relación difusa rel en $X \times Y$. En este caso el conjunto difuso B producido por "x es A" y la regla difusa "Si x es A entonces y es B" se escribe:

$$\begin{aligned} \mu_{B'}(y) &= \max_x \min[\mu_{A'}(x), \mu_{rel}(x,y)] \\ &= \vee_x [\mu_{A'}(x) \wedge \mu_{rel}(x,y)], \end{aligned} \tag{1.29}$$

O lo que es lo mismo,

$$B' = A' \circ rel = A' \circ \left(A \to B\right) \tag{1.30}$$

A continuación discutimos los aspectos computacionales del razonamiento difuso introducido en la definición anterior, y posteriormente, extender la discusión a las situaciones en las cuales se recurre a varias reglas difusas con varios antecedentes para describir el funcionamiento del sistema. No obstante restringiremos nuestras consideraciones sobre el método de implicación difusa *Mamdani*, y la composición clásica max-min por su extendida aplicabilidad la simplicidad de interpretación de sus gráficos.

El caso más sencillo es el de una sola regla con un único antecedente y tiene como formula la ecuación (1.29) Por otro lado, una regla difusa con varios antecedentes, se escribe de la forma "Si x es A y y es B entonces z es C". Y finalmente, tratamos

el caso más generalizado de varias reglas con múltiples antecedentes, que es la unión de las relaciones difusas correspondientes a las reglas difusas, por lo tanto el razonamiento difuso se puede expresar de la manera siguiente:

Permiso 1 (hecho):	x es A' y y es B'
Permiso 2 (regla 1):	si x es A_1 y y es B_1 entonces z es C_1
Permiso 3 (regla 2):	si x es A_2 y y es B_2 entonces z es C_2
Consecuencia (conclusión):	z es C'

Podemos utilizar el razonamiento difuso de la figura 1.9 como procedimiento de inferencia para calcular el conjunto difuso de salida C'.

En conclusión, podemos dividir el proceso de razonamiento difuso en cuatro pasos.

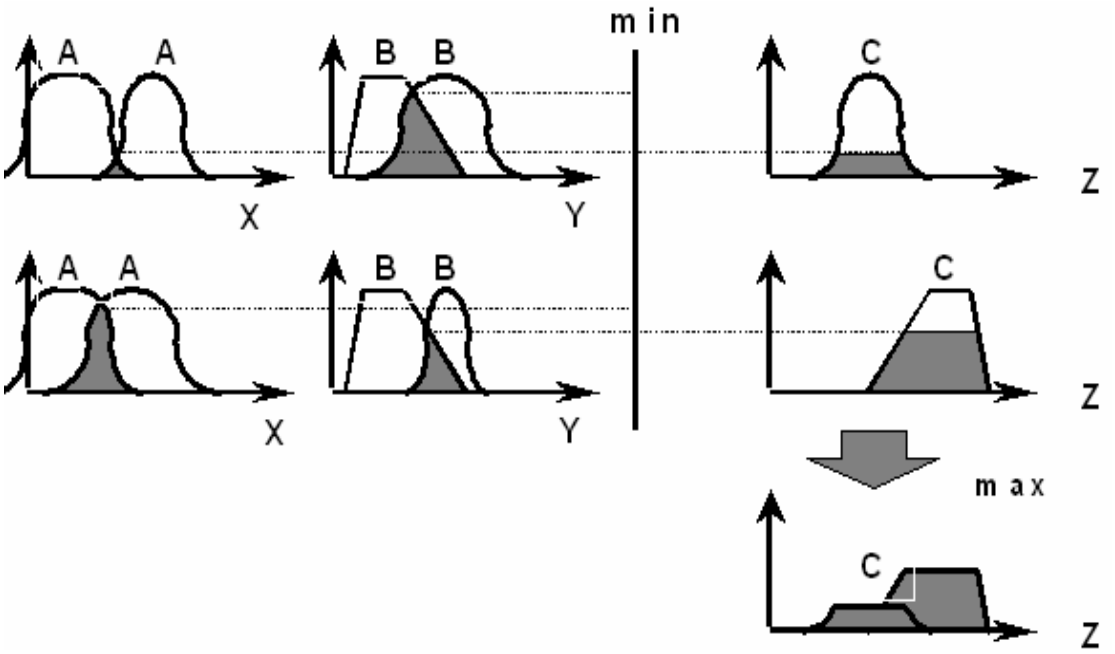


Figura 1.9: *Razonamiento difuso con varias reglas de varios antecedentes.*

Grados de compatibilidad comparar los hechos con los antecedentes de las reglas difusas para encontrar los grados de compatibilidad con respeto a función de pertenencia antecedente

Cumplimiento de antecedentes combinar los grados de compatibilidad con las funciones de pertenencia en las reglas que contienen operadores "*Y*" o "*O*" para definir que parte del antecedente se cumple.

Función de pertenencia de salida global agregar todas las consecuencias cualificadas para obtener una función de pertenencia de salida global.

1.3.5. Sistemas de inferencia difusa

Un Sistema de inferencia difusa es la estructura de computación basada en los conceptos de la teoría de los conjuntos difusos, las reglas si-entonces y el razonamiento difuso. Tiene varios campos de aplicación, como el control automático, la clasificación de datos, el análisis de decisiones, los sistemas expertos, la robótica, y el reconocimiento de muestras.

1.3.5.1. Introducción

La estructura básica de un sistema de inferencia difusa puede basarse tanto en valores de entrada difusos como clásicos, y las salidas generadas son generalmente conjuntos difusos. En el caso de necesidad de una salida clásica, especialmente en situaciones en donde el sistema de inferencia difusa es un controlador, en tal caso necesitamos un método de defuzzificación para extraer los valores numéricos que mejor representan el conjunto difuso. Cuando se trata de valores de entrada y de salida numéricos el sistema de inferencia difusa implementa un esquema no-lineal del espacio de entrada al espacio de salida, este proceso se realiza a través de un numero de reglas para describir el comportamiento local de este sistema, es decir, el antecedente de una regla define una región difusa mientras la consecuencia defínela salida en la región difusa

A continuación incorporaremos algunos de los tipos de sistemas de inferencia difusa más aplicados, las diferencias entre estos sistemas reside en las consecuencias de sus reglas, por lo que el proceso de defuzzificación difiere entre uno y otro.

1.3.5.2. Modelos difusos Mamadani

El sistema Mamdani de inferencia difusa [Mamdani & Assilian 1975] fue propuesto como primer atento para controlar una maquina de vapor por un conjunto de reglas lingüísticas de control inspiradas de la experiencia humana. La figura 1.10 es una ilustración de como un sistema Mamdani de inferencia difusa con dos entradas clásicas x y y, con dos reglas, genera una salida z.

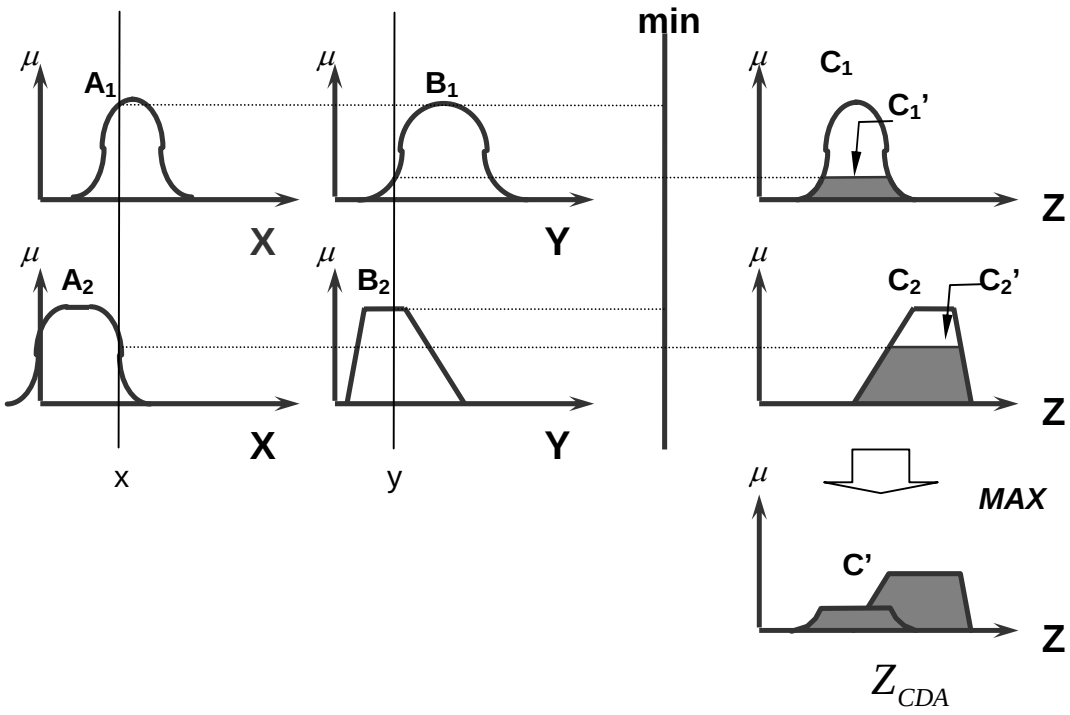


Figura 1.10: *El sistema de inferencia difusa de Mamdani usando* min *y* max *para los operadores de T-norma y T-conorma, respectivamente.*

En el ejemplo de Mamdani, se utilizó dos sistemas de inferencia difusa como controladores de la caldera y de la válvula del generador de calor, respectivamente para

regular la presión del vapor en la caldera y la velocidad del generador. Dado que la planta solo admite valores numéricos como entrada, usaremos un desfuzzificador para convertir un conjunto difuso en un valor numérico.

Defuzzificación se refiere al método con el que se extrae un valor numérico desde un conjunto difuso A dentro de un universo de discurso dado.

A continuación, una breve explicación de cada estrategia de defuzzificación.

- Centro del área Z_{CDA}:

$$Z_{CDA} = \frac{\int_Z \mu_A(z) z dz}{\int_Z \mu_A(z) dz} \tag{1.31}$$

Donde $\mu_A(Z)$ es la función de pertenencia de salida, es la estrategia de defuzzificación más adoptada, nos evoca el cálculo de los valores de las distribuciones de probabilidad.

- Bisector del área Z_{BDA}: Z_{BDA} cumple

$$\int_{\alpha}^{z_{BDA}} \mu(z) dz = \int_{z_{BDA}}^{\beta} \mu(z) dz \tag{1.32}$$

Donde $\alpha = \min\{z \mid z \in Z\}$ y $\beta = \max\{z \mid z \in Z\}$. Es decir, la línea vertical $Z = Z_{BDA}$ divide el área formado por $z = \alpha$, $z = \beta$, $y = 0$, $y = \mu_A(Z)$ en dos regiones de igual superficie.

- Media del máximo Z_{MDM}: Z_{MDM} es la media del máximo z donde la función de pertenencia alcanza un máximo μ^*. Simbólicamente,

$$z_{MDM} = \frac{\int_{Z'} z dz}{\int_{Z'} dz} \tag{1.33}$$

Donde $Z' = \{z | \mu_A(Z) = \mu^*\}$. Particularmente si $\mu_A(Z)$ tiene un máximo único $z = z^*$, entonces $z_{MDM} = z^*$. Además, si $\mu_A(Z)$ alcanza su máximo en todo $z \in \{z_{izq}, z_{der}\}$ .La media del máximo es la estrategia de defuzzificación empleada en los controladores basados en lógica difusa [Mamdani & Assilian 1975].

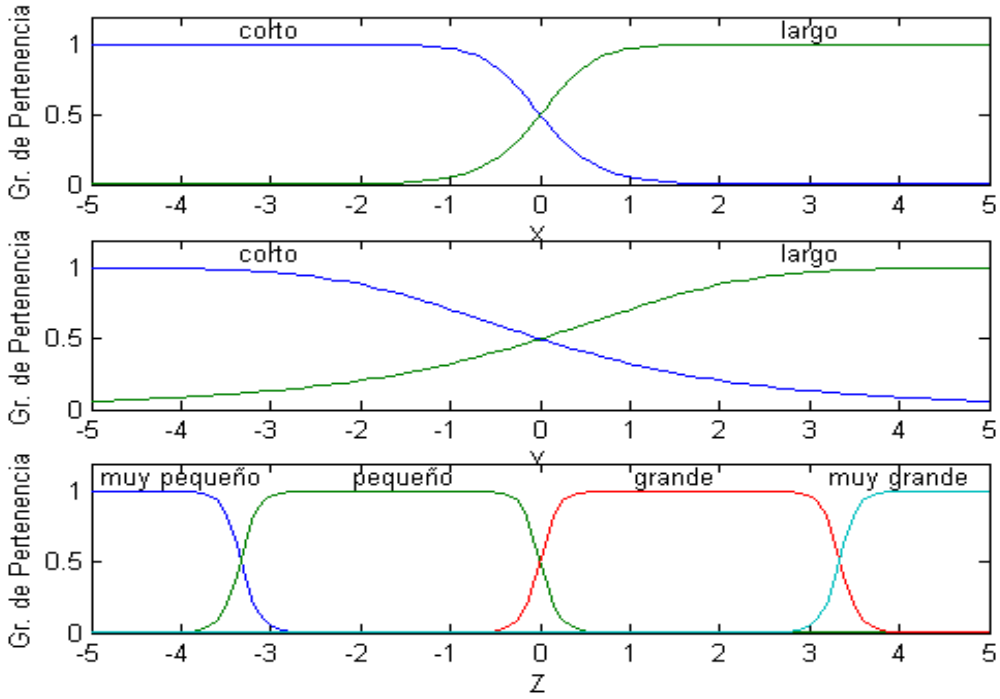


Figura 1.11: *Un modelo difuso Mamdani con 2 entradas y una salida, las funciones de pertenencia de los antecedentes y las consecuencias.*

Estas operaciones de defuzzificación no pueden, fácilmente, ser sujetas a un riguroso análisis matemático, la mayoría de los estudios se basan en los resultados experimentales. A continuación un ejemplo de un modelo difuso Mamdani con dos entradas.

Ejemplo 1.5. Modelo *difuso Mamdani con dos entradas y una salida.*

Un modelo Mamdani de dos entradas y una salida, de cuatro reglas se puede expresar:

Si X es corto y Y es corto entonces Z es muy pequeño.
Si X es corto y Y es largo entonces Z es pequeño.
Si X es largo y Y es corto entonces Z es grande.
Si X es largo y Y es largo entonces Z es muy grande.

La figura 1.11 expresa las funciones de pertenencia de las entradas X , Y y la salida Z, todos en el mismo intervalo [-5,5]. Usando el método del centro de área para la defuzzificación, sacamos la superficie global entrada – salida como muestra la figura 1.12.

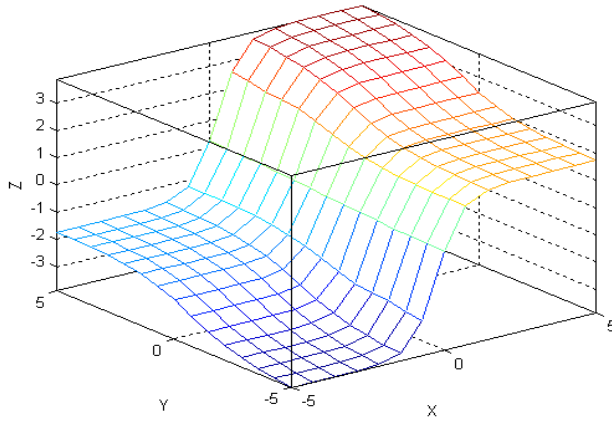


Figura 1.12: *Un modelo difuso Mamdani con 2 entradas y una salida, La superficie de salida.*

1.3.6. Conclusión

Los sistemas de inferencia difusa son las herramientas de modelado basadas en la teoría de los conjuntos difusos más importantes. Los sistemas de inferencia difusa convencionales son generalmente realizados por expertos en la materia y han sido aplicados en control automático, análisis de decisión e sistemas expertos. Las optimizaciones y las técnicas de adaptación amplían la aplicación de los sistemas de

inferencia difusa a campos como el control adaptativo, separación de señal adaptativa, regresión no-lineal y reconocimiento de modelos.

1.4. Redes Neuronales Artificiales

Las redes neuronales artificiales han sido investigadas durante más de tres décadas desde que Rosenblatt [Rosenblatt 1962] fue el primero en aplicar *perceptrones* unicapa en la clasificación de muestras. El reciente interés por el campo de las redes neuronales ha sido inspirado por las nuevas evoluciones en las redes neuronales con algoritmos aprendientes [Rumelhart 1986], los circuitos VLSI analógicos y las técnicas de procesamiento paralelo [Lippman 1987].

A continuación nos centraremos en las redes neuronales que tratan el problema del modelado usando conjuntos con los datos de entrada - salida deseados, las redes consecuentes deben de tener parámetros ajustables que puedan ser actualizados por una regla de aprendizaje supervisado.

1.4.1. El perceptrón multicapa

Definición 1.7. *Perceptrón*

> El Perceptrón representa uno de los primeros intentos de construir sistemas inteligentes y auto-aprendientes usando componentes simples, se inspira del modelo de la neurona de un cerebro biológico. [Rosenblatt 1962] diseñó su perceptrón con el objetivo de explicar y modelar las habilidades de reconocimiento de muestras de los sistemas visuales biológicos.

La primera capa de un perceptrón actúa como un detector de características especificas de las señales de entrada, la segunda capa (capa de salida), procesa las salidas procedentes de la capa de entrada y clasifica las muestras recibidas. El

aprendizaje se inicia haciendo ajustes en los pesos de conexión w_i pertinentes, y el valor del umbral θ.

La unidad de salida es un elemento umbral lineal con un valor umbral θ:

$$\begin{aligned} o &= f\left(\sum_{i=1}^{n} w_i x_i - \theta\right), \\ &= f\left(\sum_{i=1}^{n} w_i x_i + w_0\right),\ w_0 \equiv -\theta, \\ &= f\left(\sum_{i=o}^{n} w_i x_i\right),\ x_0 \equiv 1. \end{aligned} \tag{1.34}$$

Donde w_i es un peso modificable asociado a la señal entrante x_i; y $w_0 (= -\theta)$ es el bias. En la ecuación (1.34), $f(.)$ es la función de activación del perceptrón y generalmente es la función signo $sgn(x)$ o la función paso $step(x)$:

$$sgn(x) = \begin{cases} 1 & \text{si } x > 0, \\ -1 & \text{si no,} \end{cases}$$

$$step(x) = \begin{cases} 1 & \text{si } x > 0, \\ 0 & \text{si no,} \end{cases}$$

Comenzando con un conjunto aleatorio de pesos de conexión, el algoritmo de aprendizaje básico para un perceptrón unicapa repite los siguientes pasos hasta la convergencia de los pesos:

1. seleccionar una entrada x del conjunto de datos de entrenamiento.
2. si el perceptrón da una respuesta incorrecta, modificar todos los pesos de conexión siguiendo la regla:

$$\Delta w_i = \eta t_i x_i$$

Donde t_i es una salida deseada y η es el coeficiente de aprendizaje.

La regla de aprendizaje mencionada se puede aplicar para actualizar el umbral θ siguiendo la ecuación (1.34). El coeficiente de aprendizaje η puede ser una constante o una cantidad variable proporcional al error. Un η proporcional al error ayuda generalmente a una convergencia rápida pero causa un aprendizaje inestable.

Ejemplo 1.6. *El problema de O-exclusivo.*

El perceptrón elemental (unicapa) no tiene neuronas ocultas, Por lo que no puede clasificar muestras de entrada que no son separables linealmente, que es un problema muy presente a esta altura de la investigación.

Podemos considerar el ejemplo del problema del *O*-exclusivo, que trata de clasificar puntos en un *hipércubo*, cada punto de este cubo es de clase 0 o de clase 1, en nuestro caso consideramos únicamente los cuatro vértices del cuadro unidad que corresponden a las entradas (0,0), (0,1), (1,1), y (1,0). La primera y la tercera muestra son de clase 0 como demuestra:

$$0 \oplus 0 = 0$$

Y

$$1 \oplus 1 = 1$$

Donde $\oplus$ denota el operador O exclusivo. Las muestras de entrada (0,0) y (1,1) están en vértices opuestos del *cuadro unidad* aunque producen la misma salida o. Por otra parte, las muestras de entrada (0,1) y (1,0) están también en vértices opuestos pero son de clase 1, como demuestra:

$$0 \oplus 1 = 1$$

Y

$$1 \oplus 0 = 1$$

Este problema se resuelve usando una capa oculta con dos neuronas, como muestra la figura 1.13.(a), el grafico del flujo de señal en la red le muestra la figura 1.13.(b).

La neurona 1 del la capa oculta se caracteriza por:

$$w_{11} = w_{12} = +1$$

$$b_1 = -\frac{3}{2}$$

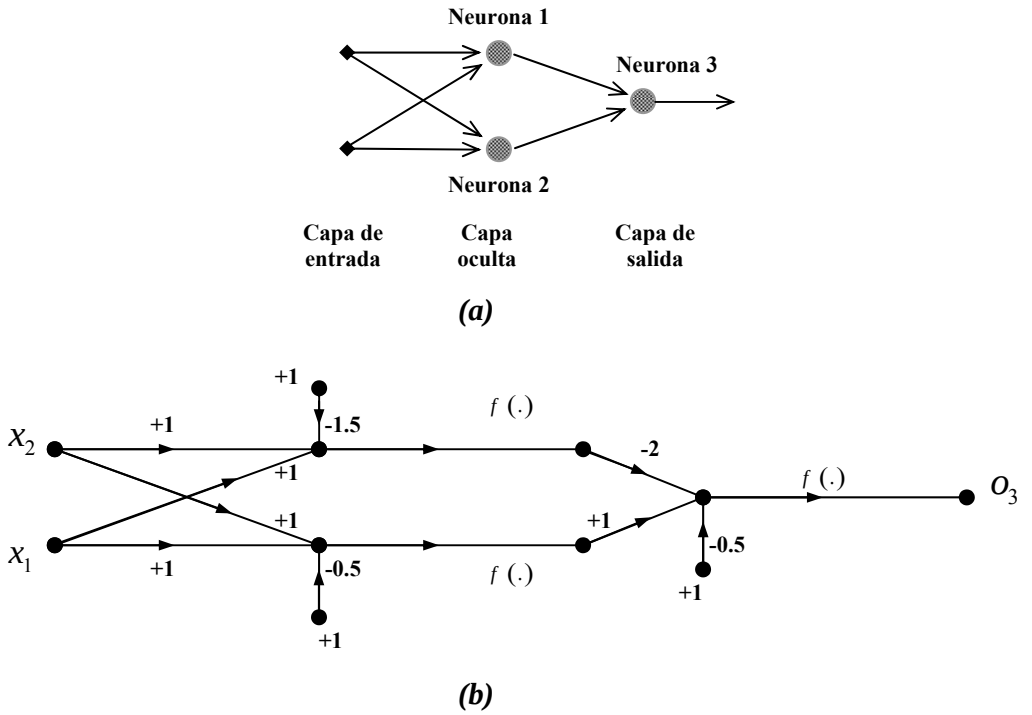


Figura 1.13: *Redes neuronales artificiales: (a) Grafico arquitectural de la red par resolver el problema del O-exclusivo. (b) Grafico del flujo de la señal de la red anterior.*

La vertiente construida por esta neurona oculta y que delimita su decisión es igual a -1, esta expresad en la figura 1.14.(a) la neurona 2 de la capa oculta se caracteriza por:

$$w_{21} = w_{22} = +1$$

$$b_2 = -\frac{1}{2}$$

La orientación y la posición de la vertiente construida por esta segunda neurona se muestran en la figura 1.14.(b)

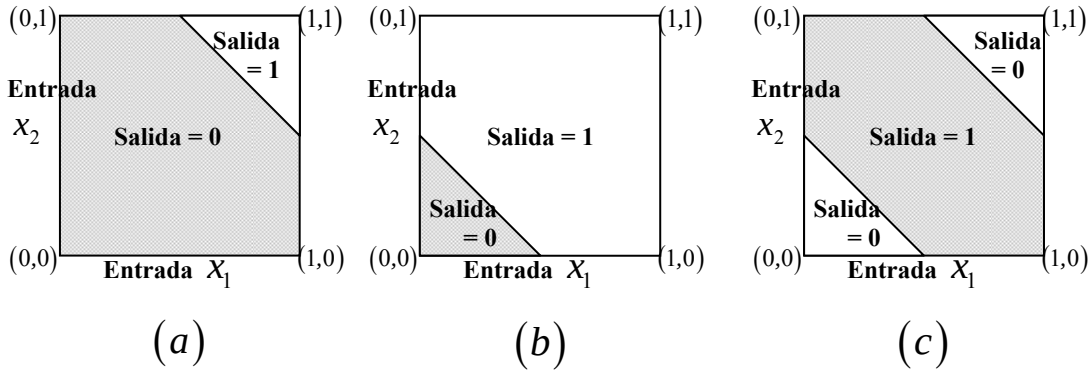


Figura 1.14: *Representación de los resultados (a) El límite de decisión de la neurona oculta 1 de la red en la figura 1.17. (b) El limite de decisión de la neurona oculta 2. (c) Los limites de decisión para la red completa.*

La neurona 3 (de salida) de la figura 1.14.(a). se caracteriza por:

$$w_{31} = -2$$

$$w_{32} = +1$$

$$b_3 = -\frac{1}{2}$$

La función de la neurona de salida es construir una combinación lineal de las vertientes de delimitación de decisión formadas por las neuronas de salida. El resultado de esta computación de muestra en la figura 1.14.(c) la neurona 2 tiene una conexión excitante (positiva) con la neurona de salida mientras la neurona 1 tiene una conexión mas fuerte e inhibitoria (negativa) con la neurona de salida. Cuando ambas neuronas están apagadas, es el caso de la entrada (0,0), la neurona de la salida permanece apagada. Cuando ambas neuronas ocultas están encendidas, el caso de la entrada (1,1), la neurona de salida se apaga de nuevo porque el efecto inhibitorio del peso negativo conectado a la neurona 1 domina el efecto excitante del peso positivo conectado a la neurona 2.

Cuando la neurona 1 esta apagada y la neurona 2 esta encendida, el caso de las entradas (0,1) y (1,0), la neurona de salida se enciende debido al efecto excitante del peso positivo de la neurona 1, de esta manera la red de la figura 1.14.(a) resuelve efectivamente el problema del *O*-exclusivo.

1.4.2. El adaline

Definición 1.8. *Adaline*

> El elemento lineal adaptativo (*ADAptive LINear Element,* Adaline) sugerido por Widrow & Hoff [Widrow & Hoff 1960] representa un ejemplo clásico del sistema auto-aprendiente inteligente más simple, es adaptable a la tarea de modelado que se le exige.

La figura 1.15. es un diagrama esquemático de esta red. Tiene una salida de red o, puramente lineal, que es una combinación ponderada de las entradas a las que se suma una constante:

$$o = \sum_{i=1}^{n} w_i x_i + w_0 \tag{1.35}$$

En una simple implementación física, las señales de entrada x_i son voltajes y w_i son conductancias de resistores controlables; la salida de la red es la suma de las corrientes provocadas por los voltajes de entrada. El problema es encontrar un conjunto de valores de las conductancias (o pesos) convenientes de manera que el comportamiento del conjunto entrada-salida del Adaline sea lo más cercano posible a los datos de entrada-salida deseados.

La ecuación del Adaline precedente es un modelo lineal exacto con $n+1$ parámetros, por lo que se puede utilizar los métodos de los mínimos cuadrados y para remediar posibles cálculos extensivos, Widrow & Hoff han introducido la regla delta para el ajuste de los pesos. Para la *p*-ésima muestra de entrada-salida, la medida del error de un Adaline con una única salida se puede expresar:

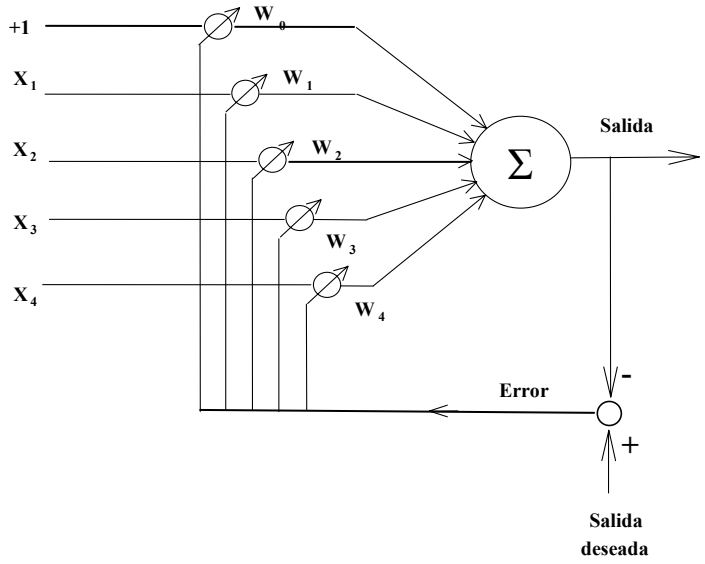


Figura 1.15: *Adaline (elemento lineal adaptativo)*

$$E_p = (t_p - o_p)^2,$$

Donde t_p es la salida deseada y o_p es la salida actual del Adaline. La derivada de E_p respeto a cada peso w_i

$$\frac{\partial E_p}{\partial w_i} = -2\left(t_p - o_p\right) x_i.$$

Por consiguiente, para disminuir E_p por el gradiente descendiente, la formula de actualización de la *p*-ésima muestra de entrada-salida es:

$$\Delta_p w_i = \eta(t_p - o_p) x_i. \tag{1.36}$$

Hay que recordar que cuando $t_p > o_p$, queremos elevar o_p aumentando $w_i x_i$; por lo tanto aumentamos w_i si x_i es positivo y disminuimos w_i si x_i es negativo. Se aplicara el mismo razonamiento cuando $t_p < o_p$.

Además de su simplicidad, la regla delta se caracteriza por un aprendizaje distribuido: se puede realizar localmente a nivel de cada nodo, ecuación (1.36).

1.4.3. Perceptrones multicapa con algoritmo de retropropagación

El perceptrón multicapa propuesto por Rumelhart [Rumelhart et al, 1986] es el componente principal de una red neuronal, proporciona la base para la mayoría de las aplicaciones de las redes neuronales [LeCun et al, 1990] [Pomerleau, 1991] [Pomerleau, 1992] [Säckinger et al, 1992] [Terrence & Rosenberg, 1986] [Terrence & Rosenberg, 1987]. Sin embargo las estrategias de aprendizaje de los primeros perceptrones unicapa con función de activación por signo o por paso no son evidentes por lo que se emplean funciones de activación continuas.

El MLP con algoritmo de retropropagación es una red adaptativa cuyos nodos (o neuronas) aplican la misma función sobre las señales entrantes**.** La figura 1.16 representa tres de las funciones de activación mas usadas en los MLP de retropropagación:

Función logística: $$f(x) = \frac{1}{1+e^{-x}}$$

Función de la tangente hiperbólica: $$f(x) = tanh(x/2) = \frac{1-e^{-x}}{1+e^{-x}}$$

Función identidad: $$f(x) = x$$

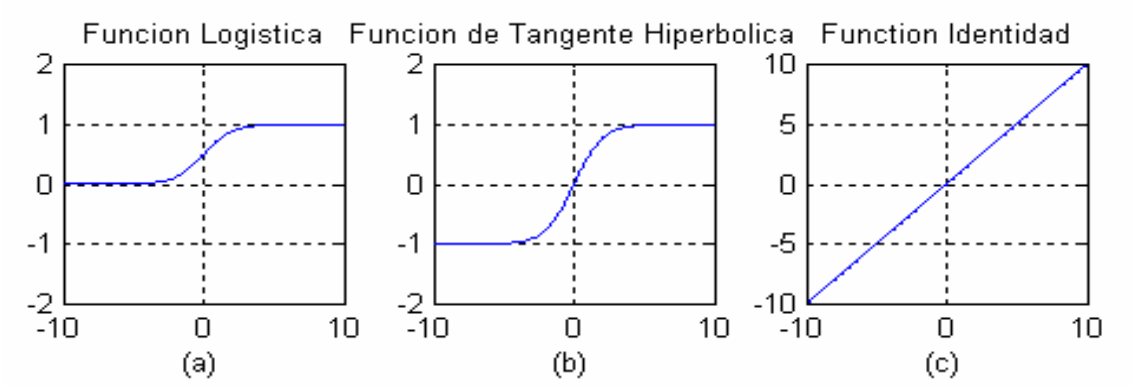


Figura 1.16: *Funciones de activación para MLPs: (a) la función logística; (b) la función tangente hiperbólica; (c) la función identidad.*

1.4.3.1. Regla de aprendizaje por retropropagación

Para simplificar, asumimos que el MLP de retropropagación utiliza la función logística como función de activación;

La entrada $\bar{x}$ de un nudo es la suma sopesada de señales entrante mas un umbral, la entrada y salida del nodo *j* en la figura 1.17. son:

$$\begin{cases} \bar{x}_j = \sum_i w_{ij} + w_j, \\ x_j = f\left(\bar{x}_j\right) = \dfrac{1}{1+\exp\left(-\bar{x}_j\right)}, \end{cases} \tag{1.37}$$

Donde x_i es la salida del nodo *i* perteneciente a cualquiera de los nodos anteriores, w_{ij} es el peso asociado al vinculo que conecta los nodos *i* y *j*, w_j es el umbral del nodo *j*, los pesos w_{ij} son parámetros internos asociados con cada nodo *j*, un cambio en los pesos de los nodos alteraría el comportamiento del nodo y por consecuencia el comportamiento del MLP de retropropagación en su totalidad. La figura 1.18 muestra un MLP de retropropagación con dos capas, tres entradas a la capa de entrada, tres neuronas en la capa oculta, y dos neuronas de salida en la capa de salida.

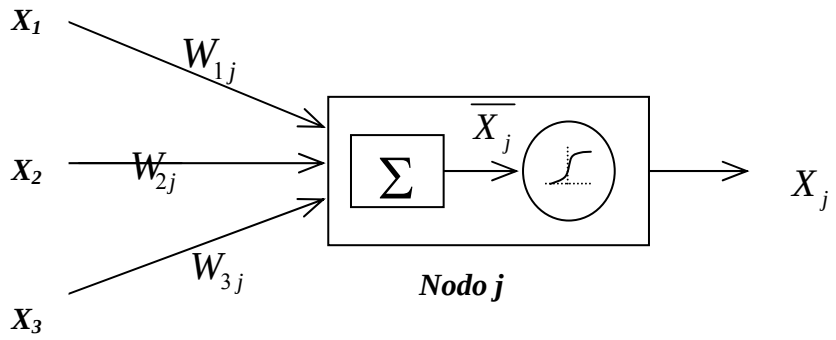


Figura 1.17: *Un nodo j de un MLP de retropropagación*

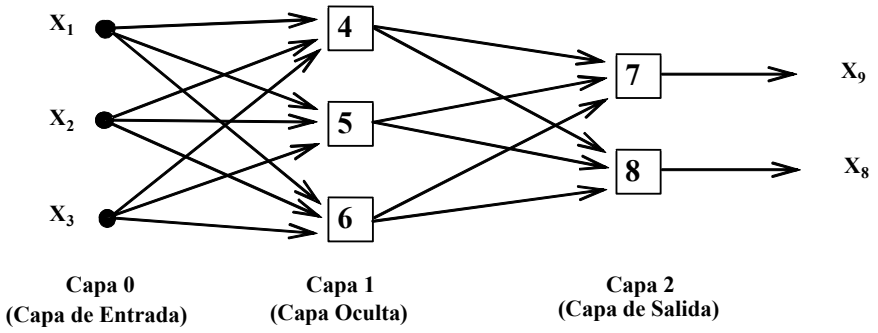


Figura 1.18: *Un MLP de retropropagación tipo 3-3-2*

Para calcular la retropropagación del error, primero definimos un error cuadrado de la *p*-ésima pareja entrada-salida:

$$E_p = \sum_k (d_k - x_k)^2 \tag{1.38}$$

Donde d_k es la salida deseada del nodo k, y x_k es la salida actual del nodo k. Para encontrar el vector gradiente, se introduce un término de error $\bar{\varepsilon}_i$ para el nodo i

$$\bar{\varepsilon}_i = \frac{\partial^+ E_p}{\partial \bar{x}_i} \tag{1.39}$$

Y la formula recursiva de $\bar{\varepsilon}_i$ se escribe

$$\bar{\varepsilon}_i = \begin{cases} -2(d_i - x_i)\dfrac{\partial x_i}{\partial \bar{x}_i} = -2(d_i - x_i)x_i(1-x_i) & \text{si el nodo } i \text{ es de salida} \\ \dfrac{\partial x_i}{\partial \bar{x}_i} = \sum_{j,i<j} \dfrac{\partial^+ E_p}{\partial \bar{x}_j}\dfrac{\partial \bar{x}_j}{\partial x_i} = x_i(1-x_i)\sum_{j,i<j} \bar{\varepsilon}_j w_{ij} & \text{si no} \end{cases} \tag{1.40}$$

Donde w_{ij} es el peso de la conexión de i hacia j; w_{ij} es nulo si no hay conexión directa. La actualización del peso w_{ki} para el aprendizaje continúo (muestra-por-muestra) es:

$$\Delta w_{ki} = -\eta \frac{\partial^+ E_p}{\partial w_{ki}} = -\eta \frac{\partial^+ E_p}{\partial \bar{x}_i}\frac{\partial \bar{x}_i}{\partial w_{ki}} = -\eta \bar{\varepsilon}_i x_k \tag{1.41}$$

Donde η es el coeficiente de aprendizaje que afecta la velocidad de convergencia y la estabilidad de los pesos durante el aprendizaje.

Para el aprendizaje discontinuo, el peso de conexión w_{ki} se actualiza tras la presentación del conjunto de datos entero o tras una época:

$$\Delta w_{ki} = -\eta \frac{\partial^+ E}{\partial w_{ki}} = -\eta \sum_p \frac{\partial^+ E_p}{\partial w_{ki}} \tag{1.42}$$

O en forma vectorial,

$$\Delta W = -\eta \frac{\partial^{+} E}{\partial W} = -\eta \nabla_{W} E \tag{1.43}$$

Donde $E = \sum_{p} E_{p}$.

1.4.3.2. Métodos para acelerar el entrenamiento de un MLP

Existen varios métodos para acelerar el entrenamiento de la retropropagación de un MLP, algunos de estos métodos se pueden aplicar a la retropropagación de la gradiente descendiente mientras otros solamente son eficientes con la retropropagación de los MLP.

Una manera de acelerar el entrenamiento discontinuo es el uso del término de ganancia:

$$\Delta w = -\eta \nabla_{w} E + \alpha \Delta w_{prec} \tag{1.44}$$

Donde w_{prec} es el valor de la actualización precedente, y la constante de ganancia α, en la práctica, toma un valor entre 0.1 y 1. La inclusión del término de prontitud estabiliza la actualización de los pesos y vigila los cambios irregulares de los pesos debidos al ruido en la gradiente o las altas frecuencias espaciales en la superficie del error.

Otra técnica práctica es la actualización normalizada de los pesos:

$$\Delta w = -\kappa \frac{\nabla_{w} E}{\left\| \nabla_{w} E \right\|} \tag{1.45}$$

Esta normalización hace que el vector del peso de la red se mueva con la misma distancia Euclidiana κ en el espacio de pesos en cada actualización, esto permite un control de la distancia κ basado en el historial de las medidas del error.

Los valores iniciales de los pesos de conexión y de los umbrales en un MLP deberían estar distribuidos uniformemente dentro de un intervalo pequeño, como [-1,1]. Si los valores iniciales de estos parámetros modificables son demasiado grandes, algunas neuronas se saturarían y producirían pequeñas señales de error, y por otra parte si el rango es demasiado pequeño, el vector gradiente seria pequeño también y el aprendizaje seria muy lento.

1.4.4. Métodos para el entrenamiento de las redes neuronales artificiales

1.4.4.1. Las redes competitivas de Kohonen

Las redes competitivas de kohonen [Kohonen 1990] son otro ejemplo de redes para clustering de datos. Las redes de este tipo imponen una restricción sobre las unidades de salida cercanas, como alguna propiedad topológica de los datos de entrada que se refleje en los pesos de las unidades des salida.

La figura 1.19.(a) representa una red competitiva de Kohonen relativamente simple con 2 entradas y 49 salidas. El proceso de aprendizaje de las mapas de Kohonen es similar al de las redes de aprendizaje competitivo, es decir, se elige una medida de similitud (o disimilitud) y la unidad ganadora es la con mayor (o menor) activación. Sin embargo en los mapas competitivos de Kohonen, no solo se actualizan los pesos de las unidades ganadoras sino también las unidades más cercanas.

Disminuye con cada iteración el tamaño del conjunto de unidades cercanas generalmente, como indica la figura 1.19(b). Una descripción secuencial de cómo entrenar una red competitiva de Kohonen viene a continuación:

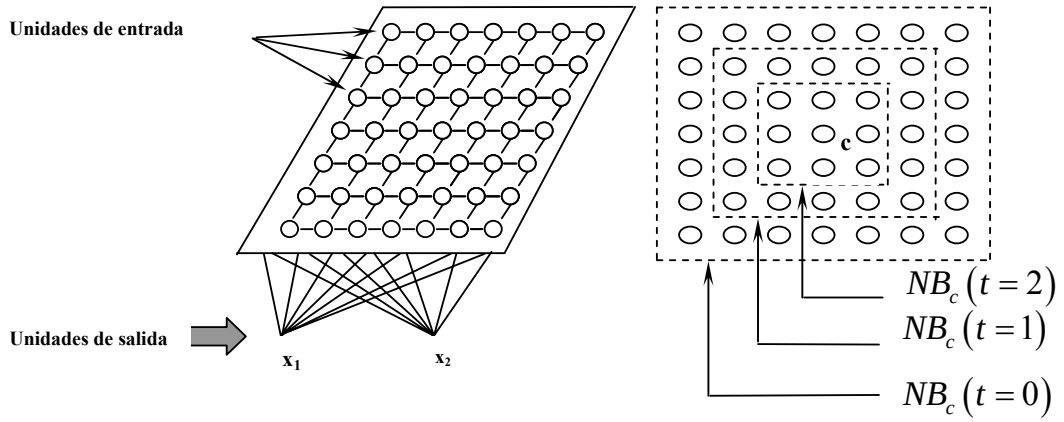


Figura 1.19: *Redes competitivas de Kohonen: (a) una red competitiva de Kohonen con 2 entradas y 49 unidades de salida; (b) la extensión de los alrededores de la unidad ganadora disminuye gradualmente con cada iteración.*

Paso 1: seleccionar la unidad con mayor medida de similitud (o menor medida de disimilitud) entre todos los vectores peso w_i y el vector de entrada x y que sería la unidad de salida ganadora. Si se elige la distancia euclidiana como medida de disimilitud, entonces la unidad ganadora *c* cumple la ecuación siguiente:

$$\|x - w_c\| = \min_i \|x - w_i\|,$$

Donde el índice *c* se refiere a la unidad ganadora.

Paso 2: sea NB_c que denota un conjunto de índices correspondientes a las unidades alrededor de c. Los pesos de la unidad ganadora y alrededores se actualizan de la manera siguiente:

$$\Delta w_i = \eta\left(x - w_i\right),\ i \in NB_c$$

Donde η es un índice de aprendizaje pequeño y positivo. A partir de la definición de los alrededores de la unidad ganadora podemos incluir una función de alrededores

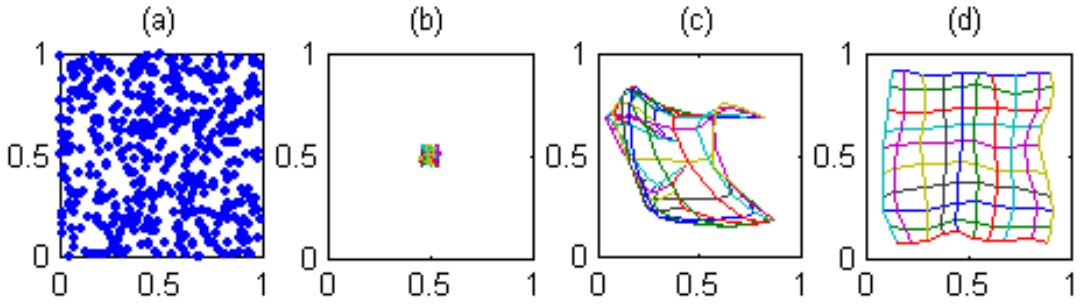


Figura 1.20: *Simulación de una red competitiva de Kohonen: (a) datos de entrada uniformemente distribuidos dentro de un intervalo* $[0,1]\times[0,1]$*; (b) los pesos iniciales; (c) los pesos después de 30 iteraciones; (d) los pesos después de 1000 iteraciones.*

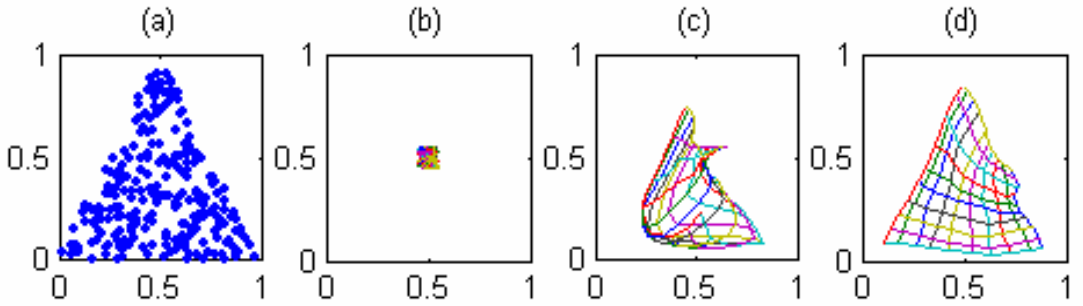


Figura 1.21: *Simulación de una red competitiva de Kohonen: (a) datos de entrada uniformemente distribuidos dentro de un triangulo; (b) los pesos iniciales; (c) los pesos después de 30 iteraciones; (d) los pesos después de 1000 iteraciones.*

$\Omega_c(i)$ alrededor de la unidad ganadora c. Se puede usar la función Gaussiana como función de alrededores:

$$\Omega_c(i) = \exp\left(\frac{-\|p_i - p_c\|^2}{2\sigma^2}\right),$$

Donde p_i y p_c son la posiciones de las unidades de salida i y c respectivamente, y σ es el radio de los alrededores. Usando la función de alrededores, la formula de actualización se rescribe de la manera siguiente:

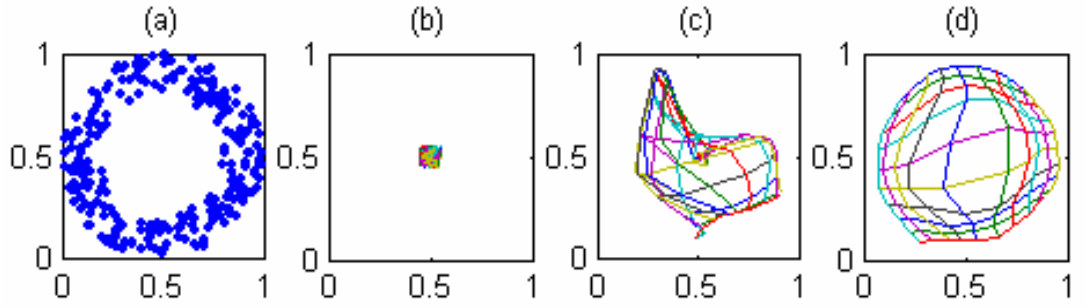


Figura 1.22: *Simulación de una red competitiva de Kohonen: (a) datos de entrada uniformemente distribuidos en forma de anillo; (b) los pesos iniciales; (c) los pesos después de 30 iteraciones; (d) los pesos después de 1000 iteraciones.*

$$\Delta w_i = \eta \Omega_c (i)(x - w_i),$$

Donde i es el índice de todas las unidades de salida.

Para obtener la mejor convergencia, habría que disminuir gradualmente el índice de aprendizaje η y el radio de alrededores σ en cada iteración, las figuras 1.20, 1.21, 1.22 presentan los resultados de una simulación de los mapas competitivos de Kohonen con diferentes distribuciones de los datos de entrada; las unidades de salida están dispuestas en una malla bidimensional. En la simulación, η y σ disminuyen linealmente respeto al numero de iteraciones.

1.4.4.2. El aprendizaje de Hebb

Hebb [Hebb 1949] describió un método sencillo de aprendizaje a partir de cambios de los pesos sinápticos, cuando dos células responden simultáneamente, sus pesos de conexión aumentan. Este fenómeno, llamado aprendizaje de Hebb, donde los pesos aumentan entre dos neuronas proporcionalmente a la frecuencia con la cual responden simultáneamente. Entre varias formulas matemáticas que representan este principio, la más sencilla se expresa:

$$\Delta w_{ij} = \eta y_i y_j, \tag{1.46}$$

Dado que los pesos están ajustados bajo una correlación entre las neuronas de salida, la formula precedente es un tipo de *regla de aprendizaje correlacional*. La salida y_i de la i-ésima neurona se puede considerar como entrada (x_i) de otra neurona j (figura 1.23) por lo que la ecuación (1.46) se puede rescribir de la forma siguiente:

$$\Delta w_{ij} = \eta y_j x_i, \tag{1.47}$$

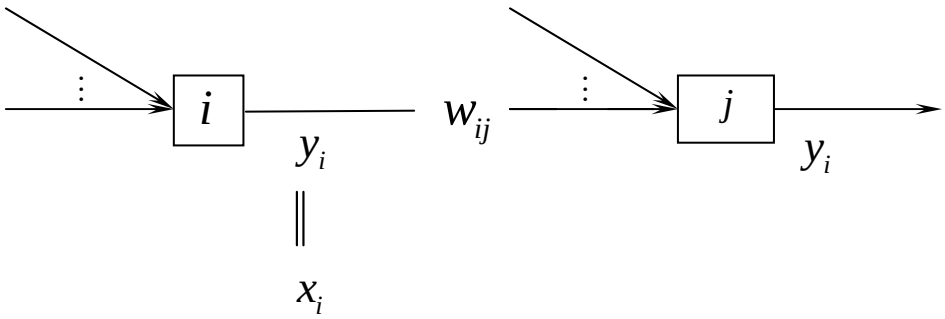


Figura 1.23: *Topología de una red con aprendizaje de Hebb; el peso w_{ij} reside entre las neuronas i y j.*

Los pesos cambian proporcionalmente a la correlación entre los señales de entrada y salida. Usando la función de neurona $f(.)$, y_i se escribe de la forma siguiente:

$$y_i = f\left(w_j^T x\right).$$

Entonces la ecuación (1.47) es equivalente a la siguiente:

$$\Delta w_{ij} = \eta f\left(w_j^T \mathrm{x}\right) x_i. \tag{1.48}$$

Creamos aquí una serie de muestras de la red de índice *p*; además, todos los pesos iniciales son *nulos*. Usando la ecuación (1.46) el valor actualizado del peso después de la presentación de todos los datos es:

$$w_{ij} = \eta \sum_{p} y_{ip} y_{jp}. \tag{1.49}$$

Las entradas frecuentes tienen mayor impacto sobre los pesos, esto hace que produzcan las mayores salidas en la red. La aplicación del aprendizaje de Hebb en la ecuación (1.46) causa una aumento espontáneo de los pesos, por lo que en algunos casos, se modifica la regla de Hebb remediar a un aumento ilimitado de los pesos.

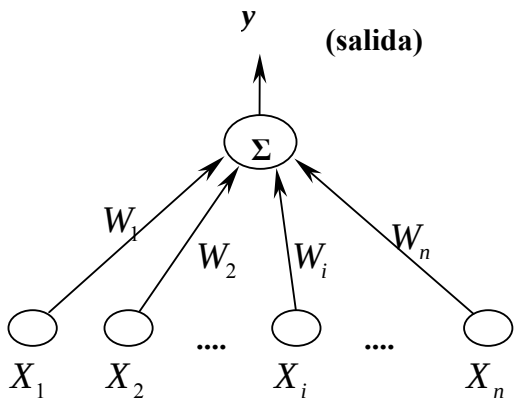


Figura 1.24: *Red de una capa y una salida, con aprendizaje de Hebb, para el análisis de componentes principales.*

Se entiende mejor el fundamento del aprendizaje de Hebb con una red neuronal de una sola entrada, n entradas y una salida, y con identificación de las funciones de activación, como muestra la figura 1.24. la salida y es igual a $\sum_{i=1}^{n} w_i x_i$, o escrito de forma matricial,

$$y = W^T X = X^T W,$$

Donde $X = [x_1, \ldots, x_n]^T$ es el vector de entrada y $W = [w_1, \ldots, w_n]^T$ es el vector peso. La regla de aprendizaje de Hebb es:

$$\Delta W = \eta y X. \tag{1.50}$$

La regla de aprendizaje precedente es un esquema de gradiente ascendiente que maximiza la función objetivo:

$$J = \frac{1}{2}\sum_p y_p^2 = \frac{1}{2}\sum_p \left(\mathrm{x}_p^T \mathrm{w}\right)^2, \tag{1.51}$$

Donde X_p y y_p son, respectivamente, la *p*-ésima entrada y su salida deseada correspondiente. Si W es un vector unidad, entonces $X_p^T W$ es la proyección del vector X_p sobre la dirección especificada por el vector W. La minimización de J implica encontrar el vector W que, con la mayor precisión, representa la *orientación* del conjunto de datos.

El aprendizaje no supervisado es muy útil en el analizar datos sin salidas deseadas; las redes evolucionan para captar las características del conjunto de datos, recordando que el aprendizaje se aplica tanto en el diseño de memorias asociativas como en problemas de optimización combinatorial. La tabla 1.3. es una breve compilación de las formulas de aprendizaje de las redes neuronales.

La mayoría de las reglas de aprendizaje tienen como formula general [Amari 1990]:

$$\Delta W_i = \eta r X, \tag{1.52}$$

Donde r es la función de aprendizaje, r puede depender o no de la disponibilidad de una salida deseada:

$r = r(\mathrm{X},\mathrm{W},\mathrm{y})$ En el caso de aprendizaje supervisado,

$r(X,W)$ En el caso de aprendizaje no supervisado.

La ecuación (1.52). corresponde a la regla de Widrow-Hoff [Widrow & Hoff 1960] cuando $r = y - W^T X$ como se describe previamente en el inicio de este. En la regla de aprendizaje de Hebb, la señal de aprendizaje r equivale simplemente a la salida de la neurona.

Algoritmo de aprendizaje	**Vector de la señal de aprendizaje**	Modo de aprendizaje
Hebb	$y\mathrm{x}$	No supervisado
Perceptrón	$\left\{t - \mathrm{sgn}\left(\mathrm{w}^T\mathrm{x}\right)\right\}\mathrm{x}$	Supervisado
LMS o Widrow-Hoff	$\left(t - \mathrm{w}^{\mathrm{T}}\mathrm{x}\right)\mathrm{x}$	Supervisado

Tabla 1.3: *Las formulas típicas de aprendizaje de las redes neuronales, con t es la salida deseada, y es la salida de la red neuronal,* ***x*** *es el vector de entrada, y* ***w*** *es el vector peso.*

1.5. Conclusión

En este capítulo hemos enumerado las principales heurísticas que forman el marco de desarrollo del algoritmo que presentamos en este trabajo, las técnicas como los algoritmos genéticos han revolucionado el mundo de la computación y constituyen una herramienta de gran utilidad para la implantación de algoritmos auto-aprendientes, y en nuestro caso y algoritmo dedicado a la aproximación de funciones.

Capítulo 2/
Diseño y optimización de RBFNs

En el capitulo anterior se han descrito las redes neuronales artificiales, que en nuestro algoritmo, procesan los datos de entrada para obtener las correspondientes salidas, estas técnicas vienen siendo aplicadas para una multitud de tareas, de las que nos interesan especialmente las que se refieren a los procesos de optimización generalmente, y muy particularmente los problemas de aproximación de funciones.

En este capitulo, se estudian de forma detallada las redes neuronales basadas en funciones de base radial, que constituyen la parte intermedia en el esquema integrado de nuestro algoritmo, se hará un repaso de los métodos que se han utilizado en la biografía para este fin.

Previamente a este paso, y en la parte inicial del algoritmo S-RBF, se ha aplicado una técnica novedosa basada en los algoritmos de clustering de datos la cual estudiamos en este capitulo.

Mediante las redes RBFN, se intenta resolver el problema de la aproximación de funciones, aplicando tanto los métodos bio-inspirados como métodos numéricos para obtener una optimización robusta, completa y eficiente.

2.1. Introducción

A Diferencia de un problema dado en el que una interpolación matricial resolvería dicho problema, en los de aproximación de funciones, esta interpolación es inaplicable en la práctica, pues supondría un coste computacional desmesurado si, a titulo de ejemplo, fijamos un número de funciones base en una red neuronal igual al de los conjuntos de datos de entrada-salida.

En los trabajos de Broomhead [Broomhead & lowe, 1988] y Moody [Moody & Darken, 1989] se obtiene un modelo basado en RBFs, para la aproximación y generalización de funciones mediante la modificación de la interpolación exacta. Este modelo proporciona una aproximación suave a los datos a partir de un número reducido de funciones base. El número de estas funciones depende más de la complejidad de la función a aproximar que del tamaño del conjunto de datos. Las principales modificaciones, respeto a una interpolación exacta, inciden en los siguientes aspectos.

1. El numero de funciones base es normalmente inferior al numero de conjuntos de datos.
2. Los centros de las funciones base se determinan como parte de un proceso de entrenamiento, en vez de en función de un conjunto de patrones de entrada.
3. Del mismo modo, y a través del proceso de entrenamiento, se ajustan los radios de cada función base, en lugar de asignar un mismo radio a todas las funciones base.
4. Se puede introducir un parámetro opcional de adaptación llamado bias para compensar una hipotética diferencia entre la media de los datos de activación de las funciones base y los correspondientes valores a aproximar.

2.2. Las Funciones de Base Radial

Las funciones de base radial, se caracterizan por una salida que se incrementa (o descrece) monótonamente con la distancia de la entrada respeto a un punto central. Son

por lo tanto funciones de la forma $R(X,u,\sigma)$, donde c es el punto central de la función R, y σ es el ancho de la misma, indica como se incrementa (o decrece) el valor de la función con la distancia de la entrada X a su centro.

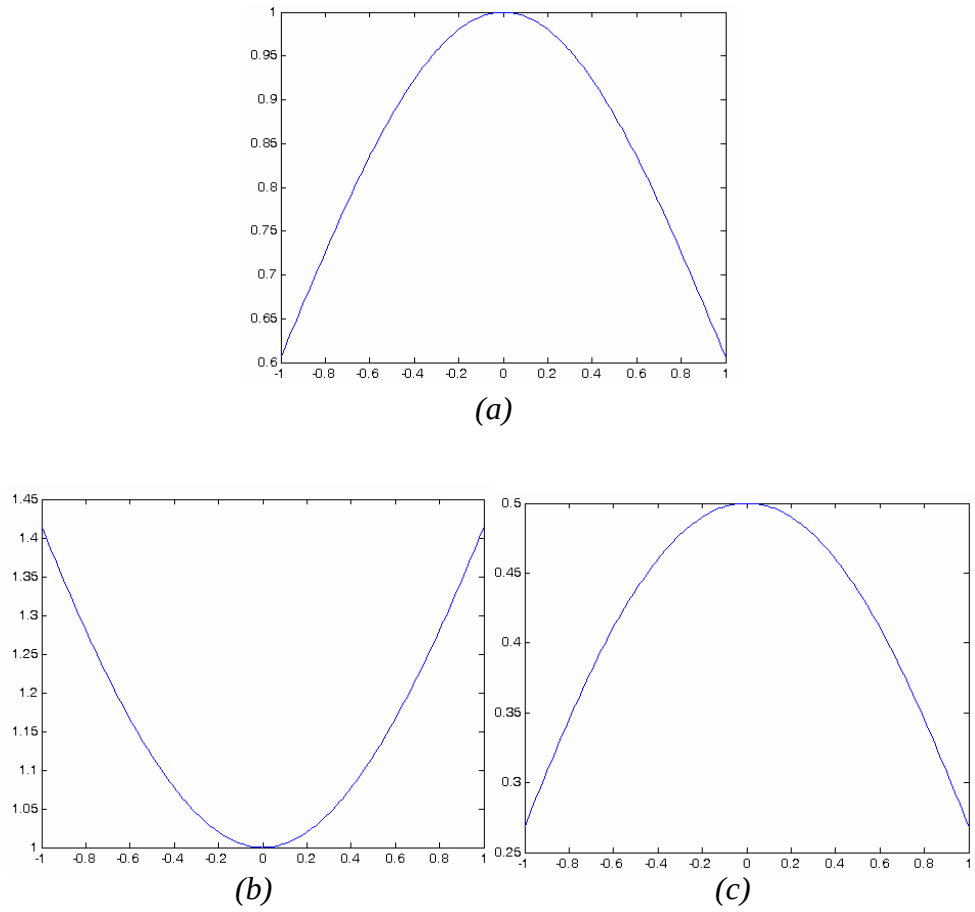


Figura 2.1: *Ejemplos de funciones de base radial centradas en 0 y de ancho con valor 1;* (a) *RBF Gaussiana:* $R(x)=\exp\left(-\|x-u\|^2/2\sigma^2\right)$*; (b) RBF cuadrática:* $R(x)=\sqrt{\|x-u\|^2+\sigma^2}$ *; (c) RBF logística:* $R(x)=\dfrac{1}{1+\exp\left[\|x-u\|^2/\sigma^2\right]}$.

La figura 2.1 muestra un ejemplo de las RBFs más utilizadas en la bibliografía. Otros ejemplos de RBFs se pueden encontrar en [Rojas et al 2000].

2.2.1. Equivalencia funcional a los sistemas de inferencia difusa

Mientras las RBFN se basan en funciones de base radial, los FIS comprenden un cierto número de funciones de pertenencia. Con aquellas funciones de forma radial, ambos FIS y RBFN tienen un mecanismo para producir una respuesta ponderada en los campos receptivos pequeños, localizando la principal excitación de la entrada. Aunque hayan sido desarrollados sobre bases diferentes, los FIS y las RBFN comparten las mismas raíces. Las RBFN manejan la convergencia rápida y los FIS se pueden desarrollar para reconocer las características de un conjunto de datos de entrenamiento de formas más efectiva que un simple MLP de retropropagación.

Los condicionantes de la equivalencia funcional entre RBFN y FIS están resumidos a continuación [Jang & Sun 1993]:

- Ambos RBFN y FIS, bajo consideraciones usan el mismo método de agregación (es decir, media ponderada o suma ponderada) para la derivación de las salida global.
- El numero de las unidades receptivas en una RBFN es igual al numero de reglas si-entonces en un FIS
- Cada función de base radial de una RBFN es igual a una MF, multidimensional compuesta, de la primera parte de una regla difusa si-entonces en un FIS. Una manera de llevarlo a cabo es usar MF Gaussianas con la misma varianza en una regla difusa, y aplicar como método para calcular el peso efectivo. El producto de esta MF Gaussianas es una función Gaussiana multidimensional – una función de base radial.
- La función de base radial y la regla difusa tendrían la misma función respuesta, Es decir tendrían los mismos términos constantes (para la RBFN y el FIS de orden cero) o ecuaciones lineales (para las RBFN extendidas y los FIS de primer orden)
- La equivalencia funcional entre los FIS y las RBFN ayuda a desarrollar ambos paradigmas de computación.

A la implementación de las funciones de base radial como funciones de activación para redes neuronales ha sido realizada por Broomhead [Broomhead & lowe 1988], han seguido posteriormente bastantes trabajos de investigación, debido a las características que tienen estas funciones como su sencilla configuración y el ser demostrado que son unos aproximadores universales como lo pueden ser los perceptrones multicapa.

2.2.1.1. Arquitecturas y métodos de aprendizaje

La figura 2.2 muestra un diagrama esquemático de una red basada en funciones de base radial con cuatro unidades receptivas; el índice de activación de la *i*-ésima unidad receptiva (o unidad oculta) es:

$$w_i = R_i(X) = R_i(\|X - u_i\| / \sigma_i),\ i = 1, 2, \ldots, H, \tag{2.1}$$

Donde X es un vector de entrada multidimensional, u_i es un vector de la misma dimensión que X, *H* es el numero de funciones de base radial (o, equivalentemente, unidades receptivas), y $R_i(.)$ es la *i*-ésima función de base radial con un máximo único en el origen. No existen pesos de conexión entre la capa de entrada y la capa oculta. Típicamente, $R_i(.)$ es una función Gaussiana

$$R_i(X) = \exp\left(-\frac{\|X - u_i\|^2}{2\sigma_i^2}\right) \tag{2.2}$$

O una función logística:

$$R_i(X) = \frac{1}{1 + \exp\left[\|X - u_i\|^2 / \sigma_i^2\right]} \tag{2.3}$$

Por consiguiente, el índice de activación de la función de base radial w_i calculado por la *i*-ésima unidad oculta es máximo cuando el vector de entrada x_i esta en el centro u_i de esta unidad.

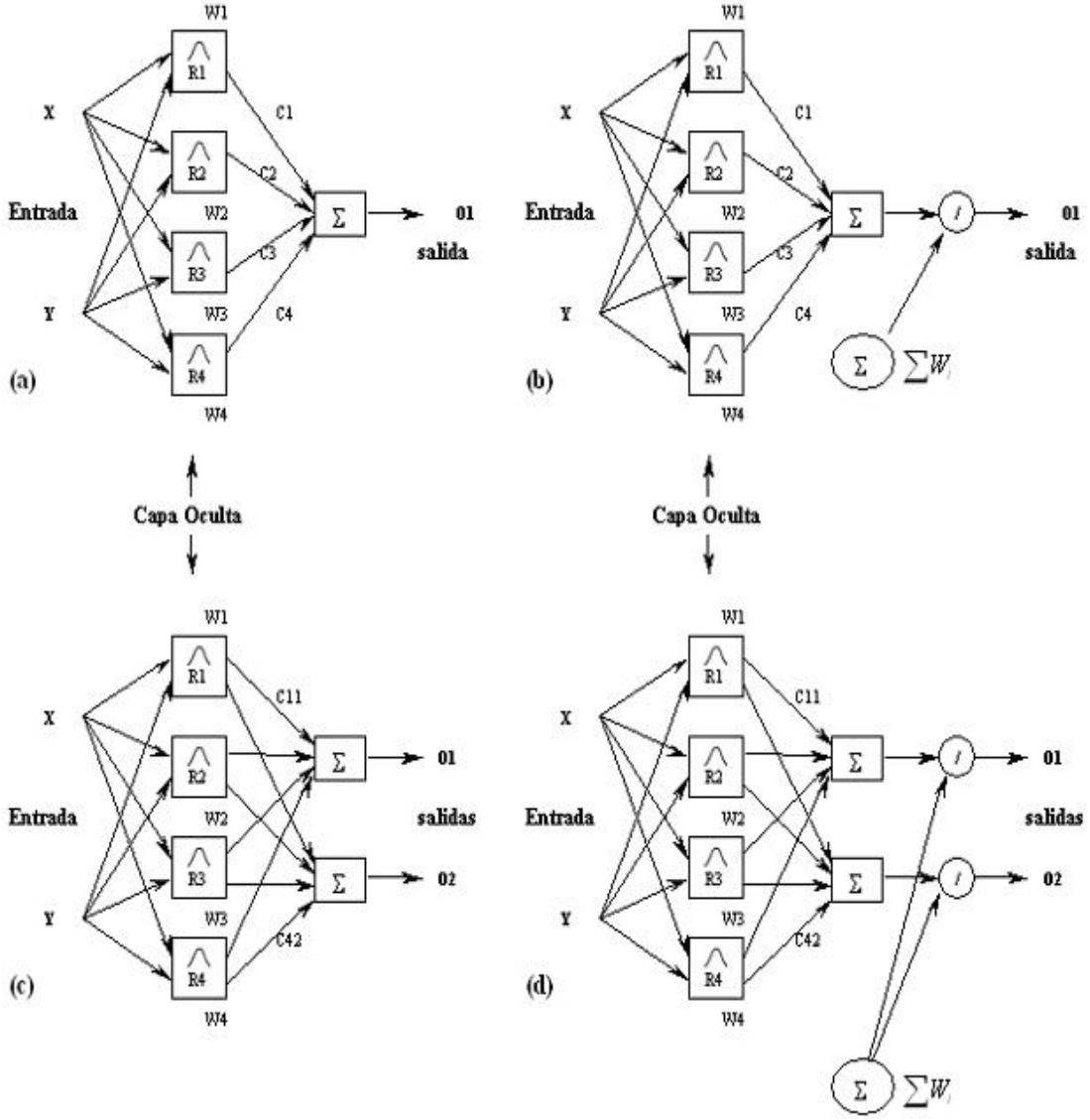


Figura 2.2: *Cuatro redes neuronales RBF procesando cuatro tipos de funciones base: (a) red RBF con salida única usando la suma ponderada. (b) red RBF con salida única usando la media ponderada. (c) red RBF con dos salidas usando la suma ponderada. (d) red RBF con dos salidas usando la media ponderada.*

La salida de una red basada en funciones de base radial puede ser calculada de dos maneras. En el método simple, como muestra la figura 2.2(a), la salida final es la suma ponderada de los valores de la salida asociados a cada campo receptivo:

$$d(X) = \sum_{i=1}^{H} c_i w_i = \sum_{i=1}^{H} c_i R_i(X) \tag{2.4}$$

Donde c_i es el valor de salida asociado al *i*-ésimo campo receptivo. Podemos también considerar c_i como el peso de conexión entre el campo receptivo i y la unidad de salida. Un método mas complicado para calcular la salida global es adoptar la media ponderada de las salidas asociadas a cada campo receptivo:

$$d(X) = \frac{\sum_{i=1}^{H} c_i w_i}{\sum_{i=1}^{H} w_i} = \frac{\sum_{i=1}^{H} c_i R_i(X)}{\sum_{i=1}^{H} R_i(X)} \tag{2.5}$$

La media ponderada, tiene un alto grado de complejidad computacional, pero es muy ventajosa en aquellos puntos de las áreas de superposición entre dos o más campos receptivos, la salida global será una interpolación entre las salidas de los campos receptivos superpuestos.

Si cambiamos la función de base radial $R_i(X)$ en cada nodo de la capa oculta en la figura 2.2(a) por su contraparte normalizada $R_i(X) / \sum_i R_i(X)$, entonces la salida global será la especificada en la ecuación (2.5). La figura 2.2 (b) da representación más explicita, donde la división de la media ponderada $\left(\sum_i c_i w_i\right)$ por la activación total $\left(\sum_i w_i\right)$ esta indicada en el nodo de división de la ultima capa. Los gráficos (c) y (d) de la Figura 2.2, equivalen a las redes RBF de (a) y (b) con de salidas en vez de una.

2.2.1.2. Interpolación y aproximación de las RBFN

Asumiendo la inexistencia de ruido en el conjunto de datos de entrenamiento, necesitamos estimar una función $d(\cdot)$ que produzca unas salidas deseadas exactas para

todos los datos de entrenamiento. Esta tarea se llama: problema de interpolación", la función resultante $d(\cdot)$ pasaría por todos los puntos de los datos de entrenamiento. Cuando usamos las RBFN con el mismo numero de funciones de base radial que el de muestras de entrenamiento, el resultado es una RBFN de interpolación, donde cada neurona de la capa oculta responde a una muestra de entrenamiento entrante particular.

Consideremos una función de base Gaussiana centrada en u_i con un parámetro de ancho σ:

$$R_i(X) = \exp\left[-\frac{(X-u_i)^2}{2\sigma_i^2}\right] \tag{2.6}$$

Cada entrada de entrenamiento x_i sirve como centro para la función base R_i, por lo tanto, de la ecuación (2.4) tenemos una RBFN de interpolación Gaussiana:

$$d(X) = \sum_{i=1}^{n} c_i \exp\left[-\frac{(X-u_i)^2}{2\sigma_i^2}\right] \tag{2.7}$$

Para un σ_i dado, con $i = 1, \cdots, n$, obtenemos las siguientes *n* ecuaciones lineales simultáneas para los *n* coeficientes de peso desconocidos c_i:

$$\begin{aligned}
d_1 &= c_1 \exp\left[-\frac{\|x_1-u_1\|^2}{2\sigma_1^2}\right] + \cdots + c_n \exp\left[-\frac{\|x_1-u_n\|^2}{2\sigma_n^2}\right] \\
d_2 &= c_2 \exp\left[-\frac{\|x_2-u_1\|^2}{2\sigma_1^2}\right] + \cdots + c_n \exp\left[-\frac{\|x_2-u_n\|^2}{2\sigma_n^2}\right] \\
&\vdots \\
d_n &= c_2 \exp\left[-\frac{\|x_n-u_1\|^2}{2\sigma_1^2}\right] + \cdots + c_n \exp\left[-\frac{\|x_n-u_n\|^2}{2\sigma_n^2}\right]
\end{aligned}$$

Escribiendo todo en forma matricial, obtenemos

$$\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} \exp\left[-\frac{\|x_1-u_1\|^2}{2\sigma_1^2}\right] & \cdots & \exp\left[-\frac{\|x_1-u_n\|^2}{2\sigma_n^2}\right] \\ \exp\left[-\frac{\|x_2-u_1\|^2}{2\sigma_1^2}\right] & \cdots & \exp\left[-\frac{\|x_2-u_n\|^2}{2\sigma_n^2}\right] \\ \vdots & \vdots & \vdots \\ \exp\left[-\frac{\|x_n-u_1\|^2}{2\sigma_1^2}\right] & \cdots & \exp\left[-\frac{\|x_n-u_n\|^2}{2\sigma_n^2}\right] \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \tag{2.8}$$

Rescribiendo la ecuación anterior de forma compacta, tenemos:

$$D = GC \tag{2.9}$$

Donde $D = [d_1, d_2, \ldots, d_n]^T$,y $C = [c_1, c_2, \ldots, c_n]^T$.

Cuando la matriz *G* es no-singular, tenemos una única solución:

$$C = G^{-1}D \tag{2.10}$$

Donde G^{-1} denota la matriz inversa de G. En la práctica sin embargo, G estará próxima a la singularidad, especialmente cuando el conjunto de entrenamiento es grande. El enfoque de regularización [Haykin 1994] permite tales casos remplazando G por $G + \lambda I$, donde λ es un numero real positivo (parámetro de regularización) y I es la matriz identidad. Poggio y Girosi [Poggio & Girosi 1990] usaron este enfoque en la implementación de una red de regularización.

Ahí donde hay menos funciones base que muestras de entrenamiento, se requiere una suposición inicial para poder determinar las posiciones de los centros, por lo que tenemos una RBFN de aproximación, en este caso, la matriz G no es cuadrada y se usan los métodos de los mínimos cuadrados para encontrar C de la ecuación (2.9).

2.2.2. Ejemplo ilustrativo: Ajustes de las RBFNs

Las figuras 2.3 y 2.4 proporcionan los resultados de un sencillo problema de interpolación con cinco puntos de datos. Se ha probado con dos RBFN de interpolación:

la RBFN con funciones de base Gaussiana en la ecuación (2.6), y una RBFN con funciones de base exponencial:

$$R_i(x) = \exp\left(-\sigma \left\| x - u_i \right\|\right) \tag{2.11}$$

Donde hemos puesto $\sigma = 1$ para ambos tipos de funciones. Los resultados presentados han sido obtenidos resolviendo la matriz de la ecuación (2.10), por lo que las curvas de interpolación resultantes pasan por todos los puntos de los datos de entrenamiento. En las figuras 2.3 y 2.4 los resultados basados en las ecuaciones (2.4) (suma ponderada) y (2.5) (media ponderada) se exponen en dúo con las cinco funciones de base radial; cuando las funciones no están lo suficiente superpuestas, la suma ponderada de las salidas ocultas basadas en la ecuación (2.4) podría generar curvas poco regulares. (Notar que las dos primeras funciones no están lo suficiente superpuestas). La normalización de la salida (media ponderada) ayuda a las funciones de base radial a cubrir el espacio de entrada para generar curvas mas regulares.

No se puede determinar que curva proporcionaría los mejores valores para los puntos intermediarios porque faltan datos sobre cualquier punto diferente de los cinco puntos de datos, sin embargo, y en términos de uniformidad, el método de la media ponderada cumple mejor que el método de la suma ponderada.

2.2.3. Comparación entre las RBFNs y los MLPs

Las RBFN y los MLP son ejemplos de redes unidireccionales no-lineales, ambos son aproximadores universales, por lo que siempre existe una RBFN capaz de imitar exactamente un MLP especificado y viceversa, sin embargo estas las RBFN y los MLP difieren en varios aspectos importantes.

1. una RBFN (en su forma básica) tiene una única capa oculta, mientras un MLP podría tener una o mas capas ocultas.
2. los nudos de computación de un MLP, situados en una capa oculta o de salida comparten un modelo neuronal común, mientras los nodos computacionales en

la capa oculta de una RBFN son diferentes y sirven para objetivos diferentes de aquellos de la capa de salida de la red.

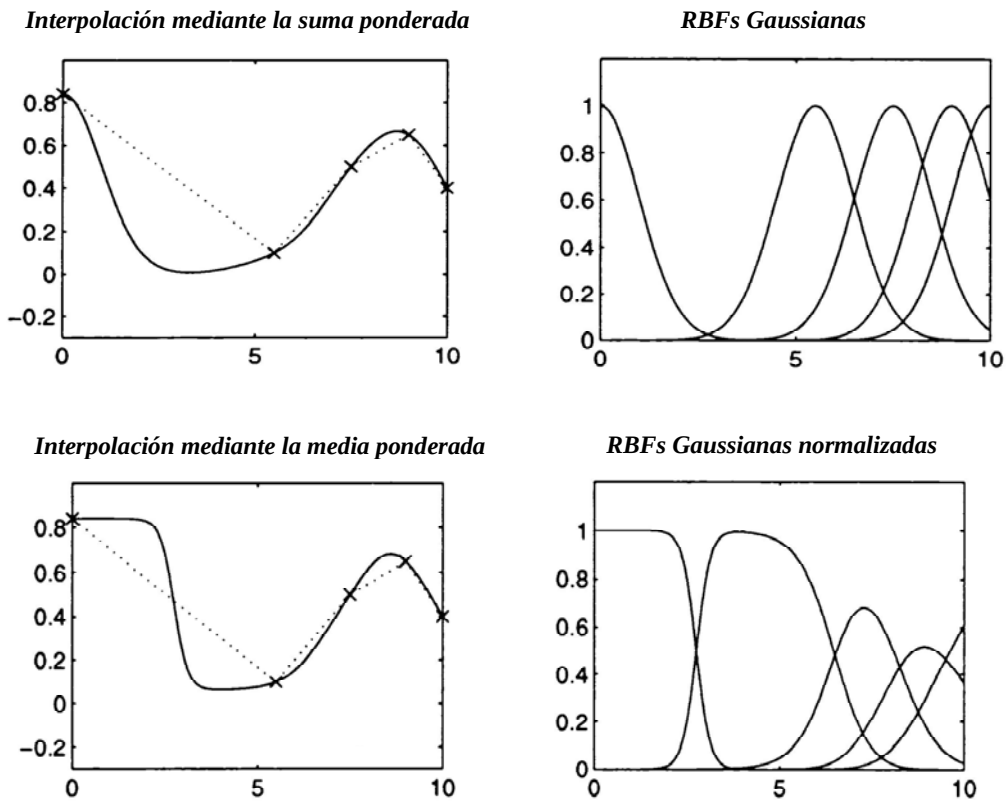


Figura 2.3: *Resultados de una interpolación mediante una red de interpolación Gaussiana.*

3. la capa oculta de una RBFN es no-lineal, mientras la capa de salida es lineal. Sin embargo las capas oculta y de salida de un MLP usadas como clasificadores de muestras son normalmente lineales. Cuando se usa el MLP para resolver problemas de regresión no-lineal, se elige una capa lineal para la salida.
4. el argumento de la función de activación de cada unidad oculta en una RBFN calcula la *norma euclidiana* (distancia) entre el vector de entrada y el centro de la misma unidad. en el caso del MLP la función de activación de cada unidad

oculta calcula el *producto interno* del vector de entrada y el vector del peso sináptico de esta unidad.

5. los MLP producen aproximaciones *globales* al trazar un mapa de entrada-salida no lineal, mientras las RBFN, usando disminuciones exponenciales (funciones Gaussianas), producen aproximaciones *locales* al trazar un mapa de entrada-salida.

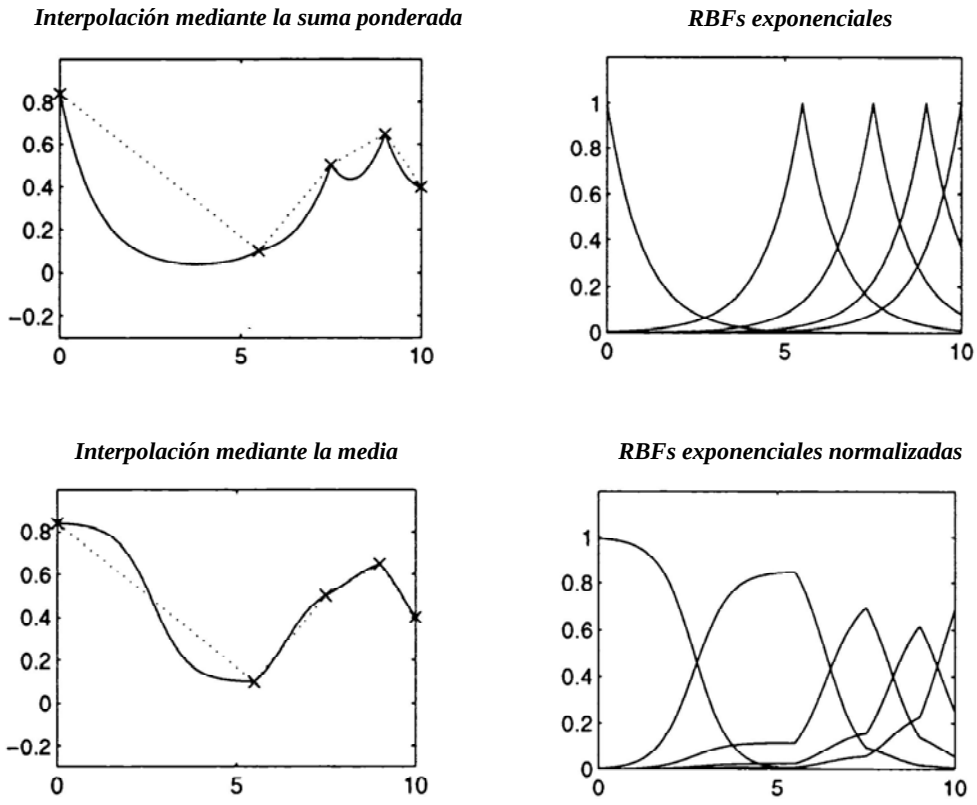


Figura 2.4: *Resultados de una interpolación mediante una red con RBFs exponenciales definidas en la ecuación (2.11).*

Esto significa que la aproximación de un trazado de entrada-salida no-lineal, el MLP requiere menos parámetros que una RBFN para un mismo grado de precisión.

Las características lineales de la capa de salida de une RBFN denotan que tal tipo de red es mas parecido al modelo de perceptrón descrito por Rosenblatt [Rosenblatt 1962] que a un MLP, sin embargo la RBFN difiere del MLP en su capacidad de implementar transformaciones lineales arbitrarias del espacio de entrada, se puede comprobar con el problema del O-exclusivo, que no es de imposible resolución usando un perceptrón lineal pero que una RBFN puede resolver.

2.3. Determinación de los parámetros de una RBFN

Las RBFNs descritas anteriormente precisan una optimización adecuada de sus parámetros con el fin de aproximar una función desconocida a partir de un conjunto de vectores de entrada-salida. El objetivo pues, reside en encontrar un conjunto de funciones base, mediante la determinación de sus centros y sus anchos, así que los pesos sinápticos asociados a estas funciones de manera que se minimice el error global de la red.

2.3.1. El error utilizado

Existen diversas estimaciones del error del aproximador para una estructura determinada. A lo largo de este trabajo, se usará el error cuadrático medio normalizado (*Normalizad Root Mean Squared Error*, *NRMSE*), puesto que al estar normalizado, permite comparar los resultados obtenidos con otras aproximaciones para el mismo problema existentes en la literatura, este error se define como:

$$NRMSE = \sqrt{\frac{\sum_{j=1}^{n}\left(y_j - f\left(x_j\right)\right)^2}{\sum_{j=1}^{n}\left(y_j - \overline{y}\right)}} \tag{2.12}$$

Donde *n* es el número de puntos que constituyen el conjunto de datos de entrenamiento formado por pares de muestras (x_j, y_j), $f(x_j)$ es la salida de la red para la *j*-ésima entrada y el valor $\bar{y}$ representa la media de las salidas deseadas y_j.

Los parámetros que definen las RBFs dentro de una red neuronal influyen de forma lineal en la salida de la misma, por lo que se utilizan algoritmos de minimización del error para realizar un ajuste iterativo hasta encontrar un mínimo óptimo. Una vez fijados los valores de estos parámetros, los valores de los pesos se pueden calcular de forma exacta los pesos de la red usando un algoritmo de resolución de ecuaciones lineales.

2.3.2. Inicialización de los centros de las RBFs: Los algoritmos de clustering de datos.

Los algoritmos de clustering han sido ampliamente aplicados, no solo en tareas de organización y categorización de datos sino también en compresión de datos y construcción de modelos tanto en el campo de los sistemas difusos como en el de las redes neuronales artificiales.

2.3.2.1. Introducción

El clustering divide un conjunto de datos en varios grupos de manera que la similitud sea mayor dentro de un grupo que entre los grupos. La realización de tal partición requiere una medida de similitud. Se consideran dos vectores de entrada para producirán valor que refleja la similitud entre ellos. Dado que las medidas de similitud son sensibles a los rangos de los elementos en los vectores de entrada, cada variable de entrada debe tener un valor normalizado en el intervalo [0,1].

El clustering es la organización, en clusters, de una colección de muestras (generalmente presentado como un vector de medidas o como un punto en un espacio multidimensional), basándose en la similitud entre estas muestras. Intuitivamente, las

muestras dentro de un cluster tienen más similitudes entre ellas que con otras muestras pertenecientes a otros clusters diferentes.

Dentro del campo de diseño de una RBFN, estos algoritmos se encuadrarían dentro de una fase de preproceso de datos de entrada de la red. Concretamente, el objetivo de esta fase es revelar la estructura interna de este conjunto de datos de entrada, agrupando o identificando grupos, de estos datos, que tengan una o varias características comunes. Una vez completada esta fase, lo normal es insertar una RBF en el centro geométrico de cada uno de los grupos identificados. La determinación de las RBFs a insertar termina con el establecimiento de un radio para cada una de ellas, relacionado con el espacio ocupado por el grupo en el que se encuentran ubicadas dichas RBFs.

En todo caso, cabe resaltar que los algoritmos de clustering no se suelen contemplar como estrategia única en el diseño de las RBFN. Así, lo normal es que estos formen parte de una fase inicial a la hora de diseñar una red de este tipo. Tal como se ha comentado, esta primera fase suele corresponder a un estudio preliminar de las características del conjunto de datos, que sirve para realizar una primera determinación o inicialización de los centros, e incluso los radios de las RBFs. De este modo, el proceso de diseño se complementaria con otras estrategias que refinan mucho más los parámetros finales de las RBFs en particular y de la RBFN en general.

Dentro de los algoritmos de clustering de datos, se puede establecer una clasificación en la que se distinguen los siguientes tipos:

- *Algoritmos de clustering no supervisados*: en este tipo de algoritmos la característica común es que el mecanismo de aprendizaje (asignación de grupos no tiene en cuenta ninguna información aportada por las variables de salida.
- *Algoritmos de clustering supervisados*: se introducen para mejorar los algoritmos de clustering no supervisados, y se caracterizan por su capacidad de tener en cuenta la información que suministren las variables de salida. Esta mejora, parece razonable, ya que las redes neuronales pretenden modelar las relaciones entre las entradas y las salidas de los

conjuntos de entrenamiento. La forma de utilizar esta información que aportan las variables de salida depende del algoritmo en si.

A continuación se describen algunos de los algoritmos de clustering de datos más importantes. De estos algoritmos, el de *c-medias*, el de *c-medias difuso* y el algoritmo *ELBG*, son algoritmos de clustering no supervisados. Por otra lado, se describe detalladamente el algoritmo CFA [González et al. 2002], especialmente diseñado para los problemas de aproximación de funciones.

2.3.2.2. Algoritmo de clustering de C-medias

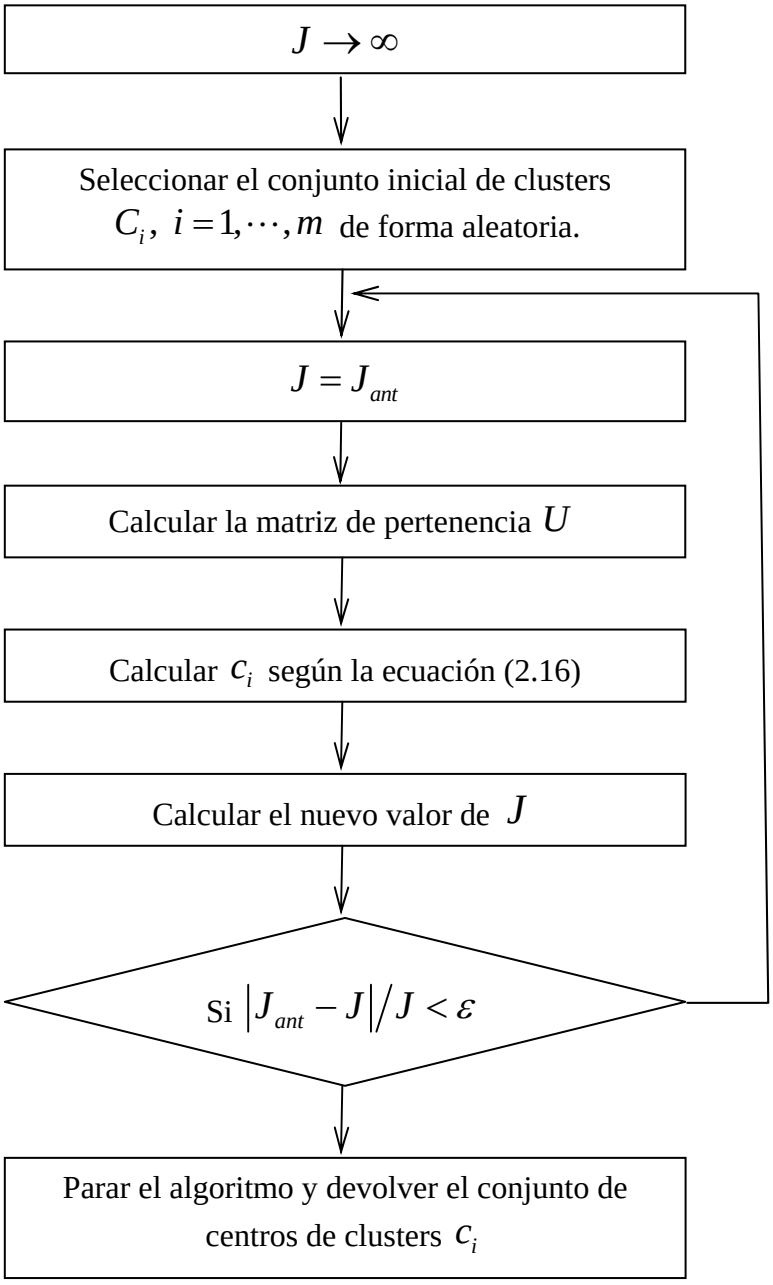
El clustering de *c*-medias [Duda & Hart 1973] ha sido aplicado en varias áreas como el procesamiento de datos para el modelado de sistemas usando RBFN [Moody & Darken 1989]. El algoritmo de *c*-medias divide un conjunto de n vectores $x_j, j=1,\ldots,n,$ en m grupos $C_i, i=1,\ldots,m,$ y encuentra un centro de cada cluster de manera que la función objetivo para medir la disimilitud está minimizada. Cuando se escoge la distancia Euclidiana como medida de disimilitud entre un vector x_k de un grupo j y su centro de cluster c_i correspondiente, la función objetivo a minimizar se puede definir de la manera siguiente:

$$J = \sum_{i=1}^{m} J_i = \sum_{i=1}^{m} \left(\sum_{k, x_k \in C_i} \left\| x_k - c_i \right\|^2 \right) \tag{2.13}$$

Donde $J_i = \sum_{k, x_k \in C_i} \left\| x_k - c_i \right\|^2$ es la función objetivo dentro de un grupo i. Vemos que el valor de J_i depende de las propiedades geométricas de C_i y de la posición de c_i.

Generalmente, se puede aplicar una función de distancia genérica $d\left(x_k, c_i\right)$ sobre el vector x_k en un grupo i; la función objetivo global correspondiente se expresa:

$$J = \sum_{i=1}^{m} J_i = \sum_{i=1}^{m} \left(\sum_{k, x_k \in C_i} d\left(x_k, c_i\right) \right) \tag{2.14}$$



Algoritmo 2.1: *Algoritmo de c-medias*

Los grupos generados por la partición están típicamente definidos por una matriz de pertenencia U binaria de dimensiones $m \times n$, donde el elemento u_{ij} es igual a 1 si x_j, el *j*-ésimo punto pertenece al grupo i y 0 sino. Une vez fijados los centros de clusters c_i, se puede escribir u_{ij} de la manera siguiente:

$$u_{ij} = \begin{cases} 1 \text{ si } \left\| x_j - c_i \right\|^2 \leq \left\| x_j - c_l \right\|^2, \text{por cada } l \neq i, \\ 0 \text{ sino.} \end{cases} \tag{2.15}$$

x_j pertenece al grupo i si, entre la totalidad de los centros de clusters, c_i es el más cercano. Puesto que un punto solo puede pertenecer a un único grupo, la matriz de pertenencia U tiene las propiedades siguientes:

$$\sum_{i=1}^{m} u_{ij} = 1, \forall j = 1, \ldots, n$$

Y

$$\sum_{i=1}^{m} \sum_{j=1}^{n} u_{ij} = n$$

Por otra parte, si se fija u_{ij}, entonces el centro óptimo c_i que minimiza la ecuación (2.14) es la media de todos los vectores del grupo i:

$$c_i = \frac{1}{\left| C_i \right|} \sum_{k, x_k \in C_i} x_k \tag{2.16}$$

Donde $\left| C_i \right|$ es el tamaño de C_i, o bien $\left| C_i \right| = \sum_{j=1}^{n} u_{ij}$.

La condición de parada del algoritmo consiste en comparar la función objetivo J en la partición actual con la de la partición anterior, y si se observa un cambio menor que un ε prefijado, se termina el algoritmo.

En modo discontinuo, el algoritmo de *C*-medias se presenta con un conjunto de datos $x_j, j = 1, \ldots, n$, el algoritmo determina los centros de clusters c_i y la matriz de pertenencia U usando los pasos siguientes en bucle:

Paso 1: inicialización del centro del cluster c_i, $i = 1, \ldots, m$, se hace generalmente seleccionando de forma aleatoria m puntos entre los puntos de datos.

Paso 2: determinar la matriz de pertenencia U usando la función (2.15).

Paso 3: calcular la función objetivo según la ecuación (2.13) parar si es inferior a un determinado valor de tolerancia o bien si su progreso respeto a la iteración previa es inferior a un determinado umbral ε.

Paso 4: actualizar los centros de clusters de acuerdo con la ecuación (2.16). Volver al paso 2.

El algoritmo es intrínsecamente iterativo, y no existe garantía de que converge hacia una solución óptima. El funcionamiento de algoritmo de *c*-medias depende de las posiciones iniciales de los centros de clusters, por ello es conveniente emplear, o bien, algún método para encontrar centros de clusters adecuados, o ejecutar el algoritmo repetidamente, probando cada vez con un conjunto de centros de clusters diferentes; por otra parte, el algoritmo precedente es únicamente representativo, es posible inicializar una matriz de pertenencia aleatoria como primer paso y seguir a continuación con proceso iterativo.

El algoritmo de *c*-medias considera como solución optima todo mínimo local cercano a la partición inicial, lo que constituye un inconveniente serio puesto que no asegura que se llegue a la partición más óptima.

2.3.2.3. Algoritmo de clustering de C- medias difuso

El algoritmo de clustering C-medias difuso es un algoritmo de clustering de datos donde cada punto pertenece a un cluster con un grado de pertenencia especificado. Bezdek propuso este algoritmo [Bezdek 1981] como mejora del anterior algoritmo de clustering C-medias duro descrito en el párrafo anterior.

El algoritmo de clustering C-medias difuso divide una colección de n vectores x_i, $i = 1, \ldots, n$ en m grupos *difusos*, y encuentra un centro de cluster para cada grupo cuya función objetivo para medir la disimilitud se trata de minimizar. La mayor diferencia

entre el algoritmo de clustering C-medias difuso y el algoritmo de clustering C-medias duro es que el primero usa repartición difusa de manera que un punto puede pertenecer a varios grupos con un grado de pertenencia especificado entre 0 y 1. Para posibilitar la repartición difusa, la matriz de pertenencia U admitirá elementos con valor entre 0 y 1, sin embargo, la normalización impone que la suma de los grados de pertenencia para un conjunto de datos sea siempre igual a 1:

$$\sum_{i=1}^{m} u_{ij} = 1, \ \forall j = 1, \ldots, n. \tag{2.17}$$

La función objetivo para un algoritmo de clustering C-medias difuso es entonces una generalización de la ecuación (2.13):

$$J\left(U, c_1, \ldots, c_m\right) = \sum_{i=1}^{m} J_i = \sum_{i=1}^{m} \sum_{j=1}^{n} u_{ij}^{p} d_{ij}^{2} \tag{2.18}$$

Donde u_{ij} es entre 0 y 1; c_i es el centro de cluster del grupo difuso i; $d_{ij} = \left\| c_i - x_j \right\|$ es la distancia Euclidiana entre el *i*-ésimo centro de cluster y el *j*-ésimo punto; y $p \in \left[1, \infty\right)$ es el exponente ponderado y nos indica el grado de difusión de la partición. Así, si $p \to 1$, la partición obtenida se aproxima más a una partición de *c*-medias convencional, mientras que si $p \to \infty$, el algoritmo produce una partición en la que todos los elementos pertenecen a todos los clusters.

Las condiciones necesarias para que la ecuación (2.18) alcanza un mínimo se pueden encontrar formando una nueva función objetivo $\overline{J}$:

$$\begin{aligned} \overline{J}\left(U, c_1, \ldots, c_m, \lambda_1, \ldots, \lambda_n\right) &= J\left(U, c_1, \ldots, c_m\right) + \sum_{j=1}^{n} \lambda_j \left(\sum_{i=1}^{m} u_{ij} - 1 \right) \\ &= \sum_{i=1}^{m} \sum_{j=1}^{n} u_{ij}^{p} d_{ij}^{2} + \sum_{j=1}^{n} \lambda_j \left(\sum_{i=1}^{m} u_{ij} - 1 \right) \end{aligned} \tag{2.19}$$

Donde λ_j, $j = 1, \ldots, n,$ son los multiplicadores de Lagrange para las n restricciones en la ecuación (2.17) diferenciando $\overline{J}\left(U, c_1, \ldots, c_m, \lambda_1, \ldots, \lambda_n\right)$ respeto a todos sus argumentos de entrada, las condiciones necesarias para que la ecuación (2.18) alcance su mínimo son:

$$c_i = \frac{\sum_{j=1}^{n} u_{ij}^{p} x_j}{\sum_{j=1}^{n} u_{ij}^{p}} \tag{2.20}$$

Y

$$u_{ij}^{p} = \frac{1}{\sum_{k=1}^{m} \left(\frac{d_{ij}}{d_{kj}} \right)^{2/(p-1)}} \tag{2.21}$$

El algoritmo de clustering C-medias difuso es un simple proceso de iteración de las dos condiciones necesarias precedentes. En modo discontinuo, el algoritmo de clustering C-medias difuso determina los centros de clusters c_i y la matriz de pertenencia Usando los pasos siguientes [Bezdek 1973]:

Paso 1: inicializar la matriz de pertenencia U con valores aleatorios entre 0 y 1 de manera que las restricciones de la ecuación (2.17) sean respetadas.

Paso 2: calcular m centros de los clusters difusos c_i, $i = 1, \ldots, n$ usando la ecuación (2.20).

Paso 3: calcular la función objetivo según la ecuación (2.18) parar si es inferior a un determinado valor de tolerancia o bien si su progreso respeto a la iteración previa es inferior a un determinado umbral.

Paso 4: calcular una nueva matriz U usando la ecuación (2.21) volver al paso 2.

Se puede también inicializar los centros de clusters.

Se puede igualmente empezar inicializando los centros de cluster y seguir con el proceso iterativo descrito anteriormente. El rendimiento del algoritmo de clustering C-medias difuso depende de los centros de cluster iniciales, por lo que nos deja un margen de utilizar otros métodos para seleccionar los centros de clusters, también se puede ejecutar el algoritmo varias veces con un conjunto de centros diferente en cada ejecución.

2.3.2.4. Algoritmo ELBG

El algoritmo ELBG pretende solucionar los problemas que presentan los algoritmos anteriores en cuanto a que sólo producen cambios locales, hasta alcanzar el mínimo local más próximo al punto de partida.

Consideremos una situación como la que se muestra en la figura 2.5, los círculos blancos indican los vectores de entrada x_j, mientras que los negros indican la disposición inicial de los centros c_i. En este caso, el centro aislado c_4 no se movería en todo el proceso puesto que no está influido por ningún vector de entrada. Por otro lado, si a la zona donde se encuentra el centro c_3 se traslada algún centro se produciría un mejor resultado ya que el conjunto de entrada quedaría mejor cubierto. El problema es que debido al funcionamiento local de los algoritmos anteriores, esto no ocurre.

Figura 2.5: *Ejemplo de centros de clusters mal colocados*

El algoritmo ELBG fue propuesto por Russo y Patané [Russo & Patané 1999] para solucionar estos inconvenientes. Este algoritmo es una extensión del de las *c*-medias y se basa en *el teorema de la distorsión total* propuesto por Gersho en [Gersho 1979], y que dice que “cada grupo hace una contribución igual a la distorsión total en una cuantización optima de alta resolución”.

Basándose en el teorema se define el concepto de utilidad de un centro de cluster c_i como:

$$p_i = \frac{J_i}{\overline{J}} \tag{2.22}$$

Donde $\overline{J}$ representa la distorsión media de la partición:

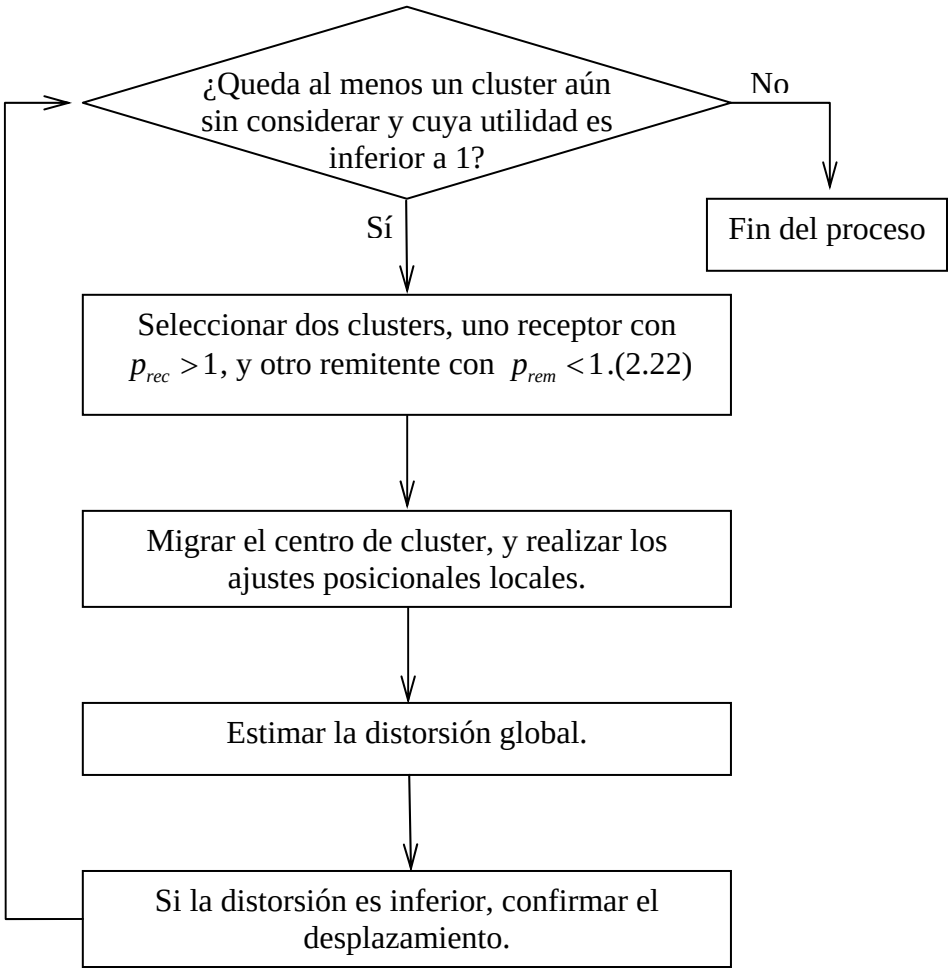
$$\overline{J} = \frac{1}{m}\sum_{i=1}^{m} J_i \tag{2.23}$$

El teorema de la distorsión total implica, que en una partición optima del conjunto de datos de entrada, todos los clusters tendrían una utilidad igual. Así, el algoritmo ELBG trata de llegar a esta condición mediante migraciones de centros con utilidad menor que 1 hacia aquellos clusters cuyos centros tienen una utilidad mayor que 1. Esto implica que, en el ejemplo de la figura 2.5, mediante el algoritmo ELBG, el centro c_4 que tiene una utilidad 0 o uno de los centros c_1 y c_2 que tienen una utilidad inferior a 1, migraría hacia el cluster cuyo centro c_3 tiene una utilidad superior a 1.

Generalmente, la migración de un cluster cuyo centro tiene una baja utilidad hacia otro más poblado se realiza a través de los pasos mostrados en el algoritmo 2.0. el primer paso consiste en calcular una posición inicial dentro del cluster de llagada, tanto para el centro que llega c_b como para el centro inicial c_a. Para esto, se considera al paralelepípedo que contiene el conjunto de vectores de entrada del cluster en cuestión, alojando c_a y c_b en su diagonal principal, de manera que cada uno esté a la misma distancia de un extremo de la diagonal, y que la distancia que los separe sea el doble. A continuación, y mediante un algoritmo de *c*-medias convencional, se ajustan c_a y c_b de manera a minimizar la función objetivo del cluster en cuestión.

Acabado el proceso de migración, se comparan las distorsiones de los grupos afectados por este desplazamiento, si se observa una minimización de la función objetivo, la migración queda aprobada y sino, se anula el desplazamiento.

Este algoritmo resuelve los inconvenientes que presentan los algoritmos anteriores, detectando los centros de clusters menos eficientes y migrándolos hacia aquellas zonas del espacio de entrada con mayor densidad en vectores de datos de entrenamiento.



Algoritmo 2.2: *Descripción detallada del proceso de migración en el algoritmo ELBG.*

2.3.3. Algoritmos específicos para la inicialización de los centros de RBFs: el algoritmo CFA.

El algoritmo que se describe a continuación es el que ha sido elegido para determinar los centros de clusters, dentro del esquema del algoritmo S-RBF. El algoritmo CFA (*Clustering for Function approximation*), es un nuevo concepto de Clustering especialmente concebido para la aproximación de funciones, ya que los algoritmos convencionales de clustering generalmente suficientes para resolver problemas de clasificación, son incapaces o insuficientes de hacerlo en el caso de la aproximación de funciones. El espacio de salida es bastante diferente entre caso y otro, siendo el de la aproximación de funciones continuo e infinito mientras el de los problemas de clasificación es finito y discreto.

El mayor cambio en la técnicas de clustering para obtener una buena inicialización para los problemas de aproximación de funciones es incorporar la respuesta de la función objetivo en el proceso de clustering, esta información permite realizar una Inicialización supervisada de los centros de funciones RBF de forma que se incrementa su densidad en aquellas zonas del espacio de entrada en las que la salida de la función objetivo es más variable, en vez de en aquellas en las que haya más vectores de entrenamiento.

En los problemas de aproximación funcional mediante combinación de GA-RBFN, se pueden cubrir con una sola neurona las zonas del espacio de entrada en las que la función objetivo es constante, independientemente del numero de cromosomas que las representen en el conjunto de entrenamiento, mientras que las zonas en las que la función objetivo es más variable quedaran mejor descritas si se incrementara el numero de neuronas RBFs que las resuman.

2.3.3.1. Descripción del algoritmo

El algoritmo CFA modula la densidad de RBFs en una zona del espacio de entrada según la variabilidad de la función objetivo en dicha zona. Este algoritmo recibe como entradas:

- Un conjunto de vectores de entrada $X = (x_1, \ldots, x_n)$,
- Las salidas $y_j = F(x_j)$ esperadas para cada cromosoma del conjunto X ,
- El numero de clusters *m* en que se quiere dividir el conjunto, y que, en un algoritmo basado en RBFs, sería el número de neuronas de éste.
- El umbral de error mínimo como condición de parada del algoritmo.

La salida del algoritmo es un conjunto de m vectores $c_1, \ldots, c_m$, cada uno representando un cluster de la partición del conjunto X realizada por el algoritmo.

Como el resto de los algoritmos de clustering, el algoritmo CFA revela la estructura subyacente de los datos en el espacio de entrada, pero de forma que se preserve la homogeneidad de las salidas de los ejemplos pertenecientes a un mismo cluster. Para llevar a cabo esta tarea, el algoritmo CFA incorpora un conjunto $O = (o_1, \ldots, o_m)$ que representa una estimación de salida esperada para los puntos que pertenezcan a un mismo cluster. El valor de o_i se obtiene como una media ponderada de las salidas de los ejemplos de entrenamiento pertenecientes al cluster C_i. La función de distorsión a minimizar en este caso se define como:

$$J = \frac{\sum_{i=1}^{m} \sum_{k, x_k \in C_i} \left\| x_k - c_i \right\|^2 \omega_{ki}}{\sum_{i=1}^{m} \sum_{k, x_k \in C_i} \omega_{ki}} \tag{2.24}$$

Donde ω_{ki} pondera la influencia de cada vector de entrenamiento x_j en la posición final del *i*-ésimo vector de centros (los centros de clusters se aplican a los centros de RBFs). Matemáticamente, ω_{ki} se define como:

$$\omega_{ki} = \frac{\left| F(x_k) - o_i \right|}{\max_{j=1}^{n} \left\{ F(x_j) \right\} - \min_{j=1}^{n} \left\{ F(x_j) \right\}} + u_{\min},\ u_{\min} > 0 \tag{2.25}$$

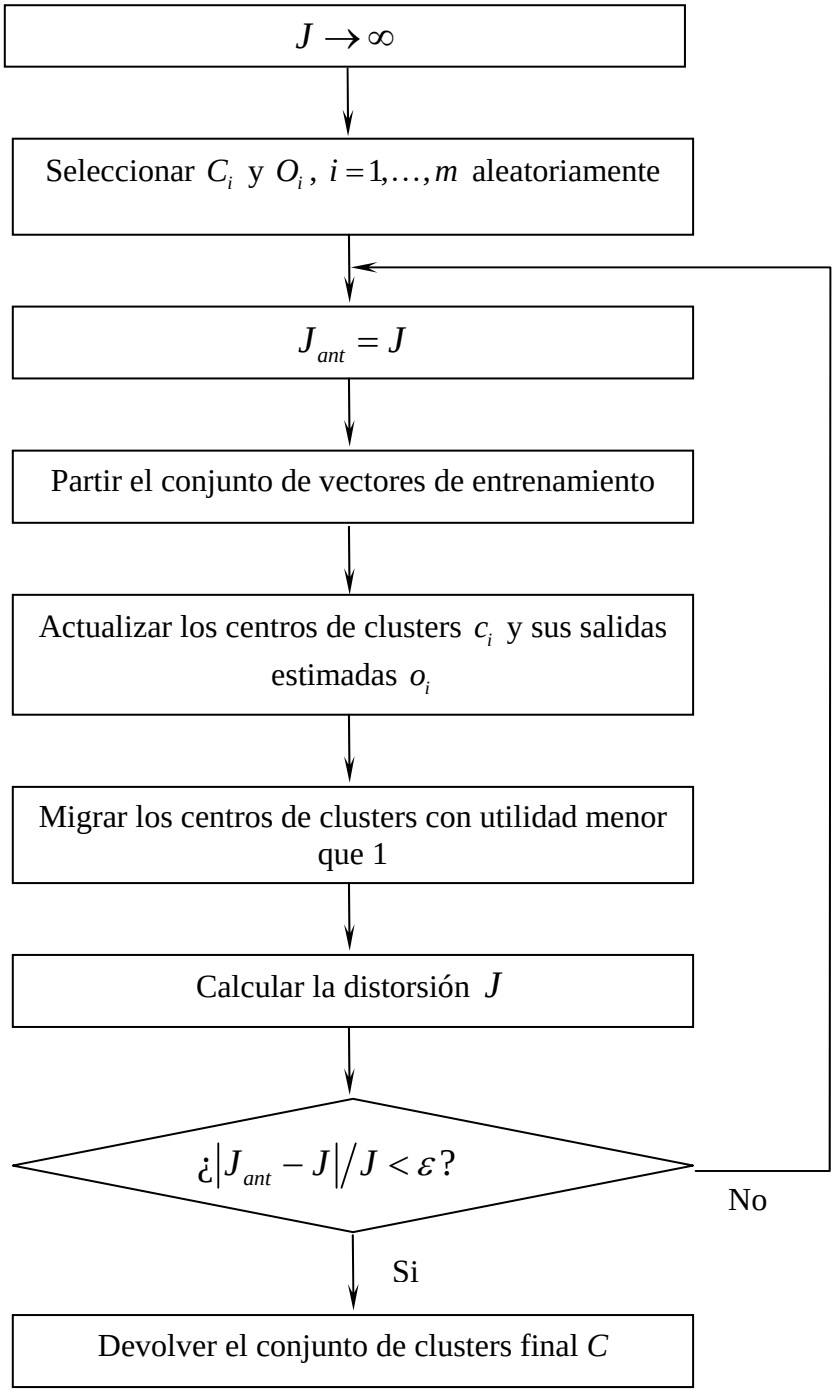


Figura 2.3: El a*lgoritmo CFA.*

El primer sumando de esta expresión calcula una distancia normalizada entre $F(x_k)$ y o_i, mientras que el segundo sumando es un umbral de mínima contribución. A medida que el valor de $u_{\min}$ decrece, el algoritmo CFA fuerza a los vectores de centros a concentrarse en las zonas del espacio de entrada en las que la función objetivo es más variable. Esta acción preserva la homogeneidad en la salida de los ejemplos x_k pertenecientes a un mismo cluster C_i, ya que se está dando más importancia a la distancia entre $F(x_k)$ y o_i en la función de distorsión. Por otro lado si el valor de $u_{\min}$ crece, el algoritmo CFA empieza a dar más importancia a la distancia entre x_k y c_i, adquiriendo el comportamiento de un algoritmo de clustering convencional.

La organización básica de un algoritmo CFA consiste, como se muestra en el algoritmo 2.3, en la iteración de tres pasos: la partición del conjunto del puntos de entrenamiento, la actualización de los centros de clusters y sus salidas estimadas, y la migración de centros de clusters desde clusters con una distorsión baja hacia clusters con una gran distorsión.

2.3.3.2. Partición de los vectores de entrenamiento

La partición de los vectores de entrenamiento se realiza mediante la aplicación de la siguiente función de pertenencia:

$$u_{ij} = \begin{cases} 1 \text{ si } \left\| x_j - c_i \right\|^2 \leq \left\| x_j - c_l \right\|^2, \text{por cada l} \neq i, \\ 0 \text{ sino.} \end{cases} \tag{2.26}$$

Esta función de pertenencia define una partición del conjunto de vectores de entrada en *m clusters* de la forma:

$$X = \bigcup_{i=1}^{m} C_i \tag{2.27}$$

Donde

$$C_i = \left\{ x_j \in X : u_{ij} = 1 \right\}, \qquad i = 1, \ldots, m \tag{2.28}$$

2.3.3.3. Actualización de los centros de clusters y sus salidas esperadas

Hasta obtener alcanzar la convergencia, la actualización de los centros de clusters sigue un proceso iterativo que actualiza c_i y o_i como las medias ponderadas de los datos de entrenamiento pertenecientes a cada cluster, y de las respuestas de sus salidas.

Las expresiones usadas para actualizar c_i y o_i son:

$$c_i = \frac{\sum_{k, x_k \in C_i} x_k \omega_{ki}}{\sum_{k, x_k \in C_i} \omega_{ki}} \tag{2.29}$$

$$o_i = \frac{\sum_{k, x_k \in C_i} F\left(x_k\right) \omega_{ki}}{\sum_{k, x_k \in C_i} \omega_{ki}} \tag{2.30}$$

La actualización de los centros de clusters es un proceso iterativo porque c_i y o_i son interdependientes, para actualizar c_i se usa el valor anterior de o_i, por lo tanto se repite el proceso hasta lograr que c_i converja hacia su posición correcta dentro del cluster C_i, cumpliendo con la condición de parada.

2.3.3.4. Migración de los centros de clusters

Puesto que la iteración de las particiones y de las actualizaciones solamente mueven localmente los centros de clusters, el algoritmo CFA incorpora un paso de migración aleatoria para evitar la convergencia hacia mínimos locales. Este proceso desplaza a los centros de clusters que se encuentran en zonas del espacio de entrada en las que la función objetivo sea estable hacia zonas con una variabilidad alta en la salida. Haciendo que el resultado final sea independiente de la configuración inicial. Para detectar dichas zonas, el proceso de migración se basa en el concepto de utilidad de un centro, definida como:

$$p_i = \frac{J_i}{\overline{J}} \tag{2.31}$$

Donde J_i es la contribución del *i*-ésimo cluster a la distorsión total:

$$J_i = \frac{\sum_{k,x_k \in C_i} \left\| x_k - c_i \right\|^2 \omega_{ki}}{\sum_{i=1}^{m} \sum_{k,x_k \in C_i} \omega_{ki}} \tag{2.32}$$

Y $\overline{J}$ se define como la distorsión media dentro de un cluster:

$$\overline{J} = \frac{1}{m} \sum_{i=1}^{m} J_i = \frac{J}{m} \tag{2.33}$$

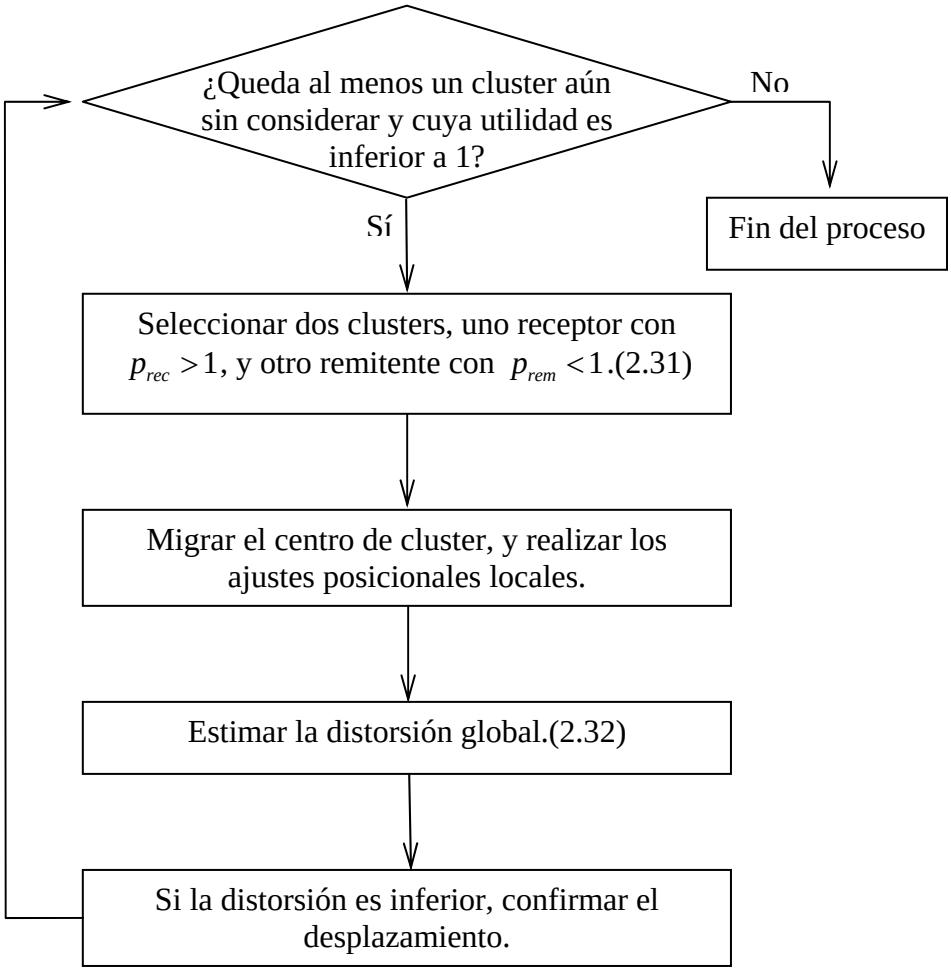
El proceso de migración, como muestra el algoritmo 2.4, desplaza a los centros de clusters c_i con utilidad menor que 1 hacia clusters con utilidad mayor que 1. Cuanto mayor sea la utilidad de un cluster, más probabilidad tendrá de ser seleccionado para recibir a un centro con utilidad menor que 1.

2.3.3.5. Ejemplo de convergencia del algoritmo CFA

Consideremos la función objetivo:

$$f(x) = \frac{\sin(2\pi x)}{\exp(x)} \qquad x \in [0,10] \tag{2.34}$$

La figura 2.6 muestra un ejemplo de migración del centro c_a hacia el cluster C_b que afecta a la totalidad de los clusters. Tras esta operación, hay que realizar los ajustes pertinentes en los clusters afectados, así pues, se inicializa un algoritmo de *c*-medias parcial al cluster C_b dividiéndolo en dos clusters C'_a y C'_b. Los vectores del cluster C_a son cedidos al cluster más cercano C_c. Una vez creado los nuevos clusters, se ejecuta el proceso de actualización, para obtener los centros y las salidas estimadas. Al final, se admite el desplazamiento si justifica una distorsión menor a la anterior.



Algoritmo 2.4: *Descripción detallada del proceso de migración en el algoritmo CFA.*

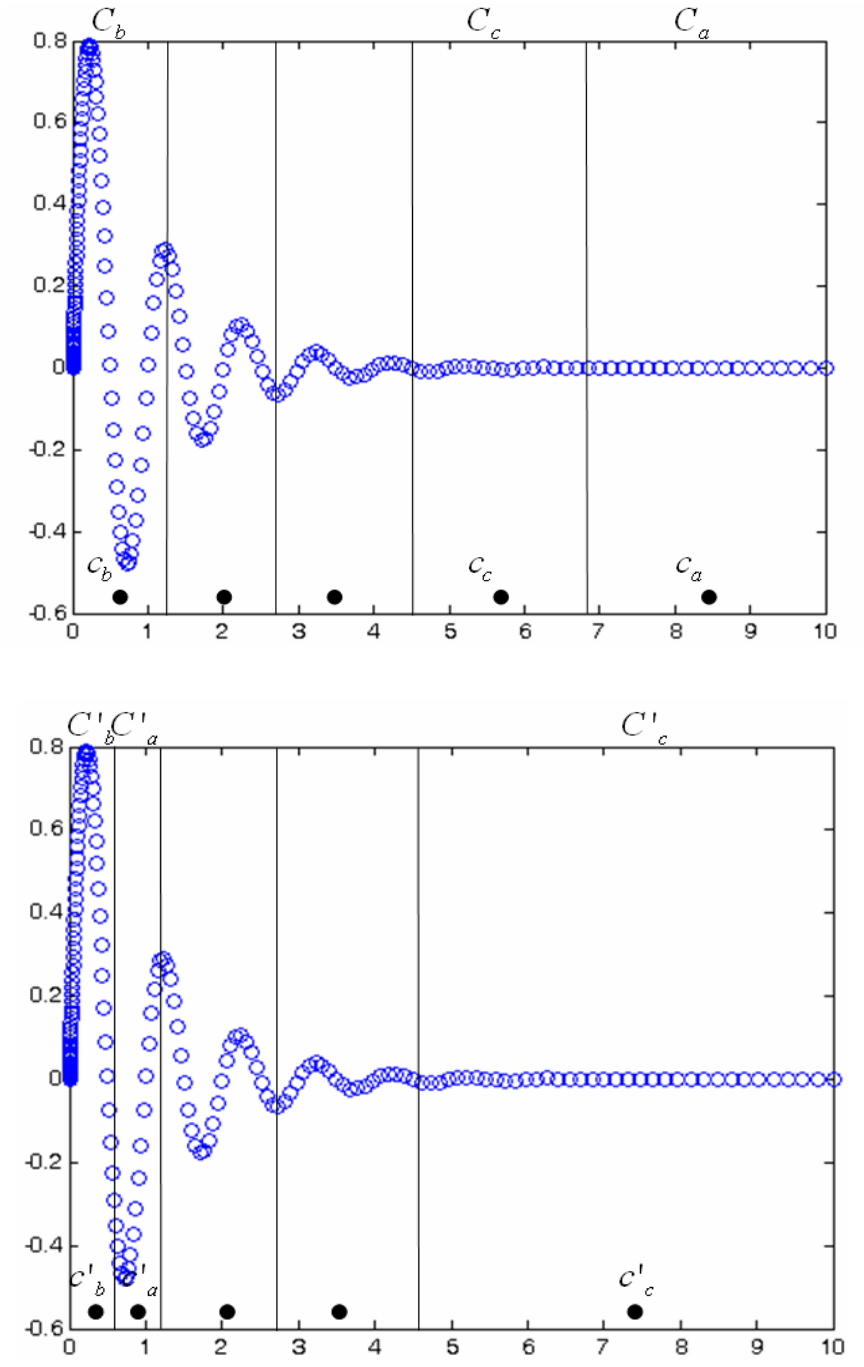


Figura 2.6: *Migración de los centros de clusters: (a) situación inicial; (b) Después de la migración del centro* c_a *hacia el cluster* C_b.

El parámetro $u_{\min}$ controla la forma en que el algoritmo CFA trata a la variabilidad de la función objetivo. Un valor alto de $u_{\min}$ implica que el algoritmo preste más atención al espacio de entrada que al de salida, y conforme su valor va bajando, la variabilidad de la función objetivo empieza atenerse más en cuenta.

2.3.4. Algoritmos para la inicialización de los anchos de las RBFs

Una vez inicializados los centros de las RBFs, el siguiente paso es encontrar los valores iniciales de los anchos (radios) de estas funciones. La condición básica en este caso es que realizar una cobertura inicial conveniente del espacio de entrada. Existen varias técnicas apropiadas para esta tarea, que mayormente utilizan la distancia como unidad de medida. Del mismo modo que se recurre a estas técnicas para la fase de inicialización, también se utilizan como referencia cuando, en fases posteriores, se introducen RBFs para las que hay que establecer un ancho. Una simple solución consiste en establecer el mismo ancho para todas las RBFs, está probado en [Park & Sandberg 1993] que es suficiente para obtener un aproximador universal. Sin embargo, teniendo la posibilidad de establecer un ancho independiente para cada RBF, se mejoran las prestaciones del algoritmo en general [Musavi et al 1992], el objetivo es conseguir un grado de solapamiento entre las RBFs vecinas y así obtener una interpolación suave y continua en aquellas regiones del espacio de entrada que representan. Entre las heurísticas más usadas se encuentran la de *k* vecinos más cercanos (*k neighbors networks, KNN*) y la técnica de la distancia media de los vectores de entrada más cercanos (*Closest Input Vectors, CIV*).

2.3.4.1. Heurística de los KNN

Mediante esta técnica [Moody & Darken 1989], se fija el radio de cada función base a un valor igual a la distancia media a los centros de sus *k* funciones base más cercanos. Esta heurística pasa a ser la heurística del vecino más cercano si se establece $k=1$, y

conforme aumenta el valor de *k*, se incrementa el grado de solapamiento entre las RBFs en la red.

2.3.4.2. Heurística de la CIV

Esta heurística [Karayiannis & Mi 1997] determina el ancho de *i*-ésima RBF mediante la siguiente ecuación:

$$\sigma_i = \frac{\sum_{k, x_k \in C_i} \left\| x_k - c_i \right\|}{k} \tag{2.35}$$

Donde *k* es el número de vectores de entrada que integran el cluster C_i. Independientemente del valor de *k*, el grado de solapamiento de esta heurística es menor a la de los KNNs.

2.3.5. Calculo óptimo de los pesos de la red

La identificación de sistemas es la interpretación matemática de un sistema dado mediante la observación de sus parejas de datos entrada-salida. Los objetivos de esta identificación son varios:

- Predecir el comportamiento de un sistema, como por ejemplo las series temporales.
- Explicar las interacciones y las relaciones entre las entradas y las salidas de un sistema.
- Para realizar simulaciones controladas de un sistema dado se necesitan un modelo de este sistema

En este apartado se van a presentar unos métodos numéricos para la determinación de los pesos de conexión dentro de una RBFN de una única capa oculta, y de una salida lineal, lo que permite encara este problema como el de la resolución de un sistema de ecuaciones.

Una vez establecidos los parámetros que definen las RBFs dentro de una red neuronal, quedaría el problema de la resolución de la ecuación (2.9) teniendo en cuenta la totalidad de las muestras del conjunto de patrones.

2.3.5.1. Descomposición de cholesky

Esto implica que los métodos que se describen a continuación para determinar el vector C, son métodos para la resolución de sistemas de ecuaciones lineales. Concretamente los métodos que mas se han utilizado para esta tarea en el campo de las RBFN son la descomposición de *Cholesky* [Golub & Van Loan 1996] es el más rápido de los métodos pero solo de puede aplicar a sistemas descritos por una matriz cuadrada, simétrica y definida positiva. El sistema de ecuaciones a resolver en la ecuación (2.8) debe transformarse puesto que la matriz C no cumple estos requisitos.
Una vez realizadas las transformaciones, la matriz de activación debe verificar otra serie de condiciones cuyo cumplimiento depende de las características de las RBFs ya definidas. Así, las RBFs deberían estar colocadas de forma que se cubra todo el espacio de entrada y ser linealmente independientes. Esto implica que redes con funciones base "mal colocadas", bien porque alguna no se active con ningún punto de entrenamiento o porque exista demasiado solapamiento entre las mismas, pueden producir matrices de activación singulares que el método de Cholesky no puede resolver, o bien casi singulares, y en este caso el método anterior proporcionaría una solución indeseable.

2.3.5.2. Descomposición en valores singulares

La descomposición en valores singulares (Singular Value Decomposition, SVD) [Golub & Van Loan 1996] es un primer método para solucionar el problema de las RBFs mal colocadas algo que, por otro lado, es normal en el diseño de una RBFN. Como se ha mencionado, en estos casos se producen matrices de activación singulares o casi singulares. 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 C.

La descomposición en valores singulares facilita una solución para cualquier sistema de ecuaciones, con la que se obtiene una reducción del error en la salida de la red. Además el método muestra que, si se elimina de la red alguna función base cuyo valor singular asociado tuviera una magnitud pequeña, el error de aproximación no se vería prácticamente afectado. Al ser la red más pequeña, se gana en simplicidad y se pueden conseguir mejoras en la aproximación. Este suele ser otro uso del método SVD. Es decir, la identificación de funciones base que aportan poco al funcionamiento general de una red. El inconveniente que plantea el método es que la forma en que selecciona las funciones base más importantes de la red no tiene en cuenta la salida esperada del sistema para cada vector de entrada. De esta forma, si en una red existieran dos funciones base muy solapadas, probablemente se sacrificaría a una de las dos sin importar la influencia que dicha modificación tenga en el error de aproximación de la red. La fiabilidad del método SVD ha sido comprobada en varias áreas como el análisis estadístico [Hammarling, 1985], el procesamiento de imágenes [Andrews & Patterson, 1976], identificación de sistemas [Vandevalle & De Moor, 1988], el control de plantas [Laub, 1997] y el diseño de sistemas de inferencia difusa [Mouzouris & Mendel, 1996], [Yen & Wang, 1996], [Yen & Wang, 1999].

2.3.5.3. El método de los mínimos cuadrados

En el método de los mínimos cuadrados, utilizado en el control automático [Chen & Billings, 1992] [Chen et al, 1992], las redes neurodifusas [Wang & Mendel, 1992b], y en la predicción de series temporales, la salida de un modelo lineal Y se determina mediante una expresión lineal:

$$Y = \theta_1 f_1(X) + \theta_2 f_2(X) + \cdots + \theta_n f_n(X) \tag{2.36}$$

Donde $X = [x_1, \ldots, x_m]^T$ es el vector de entrada del modelo, $f_1, \cdots, f_n$ son funciones conocidas de X, y $\theta_1, \cdots, \theta_n$ son parámetros desconocidos a estimar. En las

estadísticas, se refiere al ajuste de datos mediante un modelo lineal como regresión lineal. Por lo que la ecuación (2.36) se llama también función de regresión, y los θ_i se denominan coeficientes de regresión.

Para identificar los parámetros desconocidos θ_i, se realizan experimentos para obtener un conjunto de datos de entrenamiento compuesto por pares de datos $\{(x_i, y_i), i = 1, \ldots, m\}$ que representan los pares deseados de entrada-salida del sistema que se quiere modelar. Sustituyendo cada par de datos en la ecuación (2.36) produce un conjunto de *m* ecuaciones lineales:

$$\begin{cases} f_1(x_1)\theta_1 + f_2(x_1)\theta_2 + \cdots + f_n(x_1)\theta_n = y_1 \\ f_1(x_2)\theta_1 + f_2(x_2)\theta_2 + \cdots + f_n(x_2)\theta_n = y_2 \\ \vdots \\ f_1(x_m)\theta_1 + f_2(x_m)\theta_2 + \cdots + f_n(x_m)\theta_n = y_m \end{cases} \tag{2.37}$$

Usando la notación matricial, podemos rescribir las ecuaciones precedentes de forma concisa:

$$F\Theta = Y \tag{2.38}$$

Donde F es una matriz de $m \times n$:

$$F = \begin{bmatrix} f_1(x_1) & \cdots & f_n(x_1) \\ \vdots & \ddots & \vdots \\ f_1(x_m) & \cdots & f_n(x_m) \end{bmatrix} \tag{2.39}$$

Θ es un vector de $n \times 1$ parámetros:

$$\Theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Y y es un vector de salida de $m \times 1$:

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

Para identificar únicamente el vector Θ, es necesario que $m \geq n$. Si F es cuadrada $(m = n)$ no-singular, entonces podemos resolver Θ en la ecuación (2.38).

$$\Theta = F^{-1}Y \tag{2.40}$$

Sin embargo, *m* es habitualmente mayor que *n*, indicando que tenemos mas pares de datos que parámetros de ajuste. En este caso, no siempre es posible una solución exacta que satisfaga todas las *m* ecuaciones, puesto que los datos podrían estar contaminados por el ruido, o que el modelo no sea el apropiado para describir el sistema objetivo. Por lo que se debe modificar la ecuación (2.38) incorporando un vector error E para explicar el ruido aleatorio o el error de modelado:

$$F\Theta + E = Y \tag{2.41}$$

Ahora, en vez de encontrar la solución exacta de la ecuación (2.38), se busca $\Theta = \hat{\Theta}$ que minimice la suma del error cuadrado definido por:

$$E(\Theta) = \sum_{i=1}^{m} \left(y_i - f_i^T \Theta \right)^2 = E^T E = \left(Y - F\Theta \right)^T \left(Y - F\Theta \right) \tag{2.42}$$

Donde $E = \left(Y - F\Theta \right)$ es el vector de error producido por una elección especifica de Θ. Notar que $E(\Theta)$ es cuadrática y tiene un mínimo único en $\Theta = \hat{\Theta}$. El teorema siguiente expone una condición necesaria del estimador de mínimos cuadrados $\hat{\Theta}$.

Teorema 1.1 *el estimador de mínimos cuadrados*

El error cuadrado de la ecuación (2.42) es mínimo cuando $\Theta = \hat{\Theta}$, llamado estimador de mínimos cuadrados (LSE), cumple la ecuación normal:

$$F^T F \hat{\Theta} = F^T Y \tag{2.43}$$

Si $F^T F$ es no-singular, $\hat{\Theta}$ es única y su forma es:

$$\hat{\Theta} = \left(F^T F \right)^{-1} F^T Y \tag{2.44}$$

existen varios métodos en la literatura para encontrar el estimador de mínimos cuadrados para la ecuación (2.38). Un enfoque sencillo es suponer que la derivada de

$E(\Theta)$ es igual a cero. Sabiendo que $\Theta^T F^T Y = Y^T F \Theta$ es un escalar, podemos ampliar $E(\Theta)$:

$$E(\Theta) = \left(Y^T - \Theta^T F^T\right)\left(Y - F\Theta\right) = \Theta^T F^T F\Theta - 2Y^T F\Theta + Y^T Y \quad (2.45)$$

Entonces la derivada de $E(\Theta)$ es:

$$\frac{\partial E(\Theta)}{\Theta} = 2F^T F\Theta - 2F^T Y \quad (2.46)$$

Poniendo $\frac{\partial E(\Theta)}{\Theta} = 0$ en $\Theta = \hat{\Theta}$, obtenemos la ecuación normal:

$$F^T F\hat{\Theta} = F^T Y \quad (2.47)$$

Si $F^T F$ es no-singular, entonces $\hat{\Theta}$ tiene una única solución:

$$\hat{\Theta} = \left(F^T F\right)^{-1} F^T Y \quad (2.48)$$

2.3.5.4. Interpretación geométrica del LSE

Sea **A** expresada como una fila de *n* vectores columna de tamaño $m \times 1$:

$$F = [\, f_1, \ldots, f_n \,]$$

Entonces tenemos:

$$F\Theta = [\, f_1 \ \ldots \ f_n \,].\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} = \theta_1[\, f_1 \,] + \ldots + \theta_n[\, f_n \,] \quad (2.49)$$

Es decir, $F\Theta$ es una combinación lineal del vector base $[\, f_1 \ \ldots \ f_n \,]$ en un espacio *m*-dimensional. Para que $F\Theta$ se aproxime a Y en el sentido de mínimos cuadrados, $F\Theta$ tiene que ser la proyección de Y en el espacio delimitado por $[\, f_1 \ \ldots \ f_n \,]$.

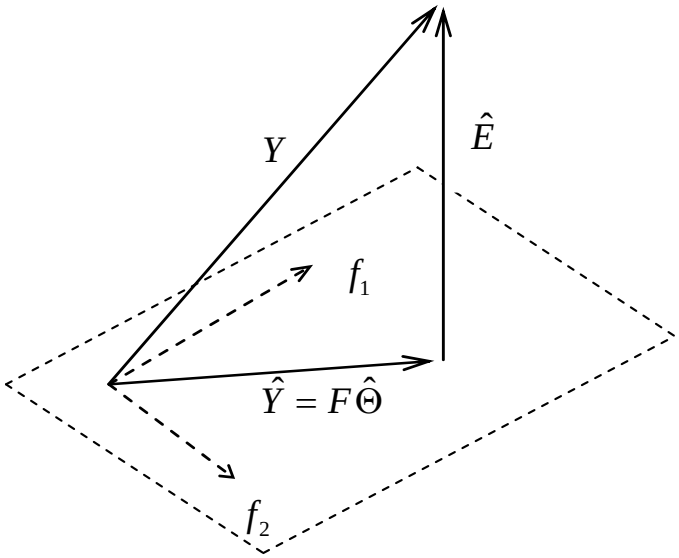


Figura 2.7: *Interpretación geométrica del estimador de mínimos cuadrados*

Para ilustrar este concepto, la figura 2.7 muestra la situación cuando $n = 2$ y $m = 3$. Tenemos $F = \left[f_1 \;\; f_2 \right]$ y $F\Theta = \theta_1 f_1 + \theta_2 f_2 . F\Theta$ está siempre dentro del espacio delimitado por f_1 y f_2. Así, para que $E = Y - F\Theta$ alcance una mínima longitud, $F\Theta$ tiene que ser igual a la proyección de Y en el espacio delimitado por f_1 y f_2, esto ocurre cuando $E = Y - F\Theta$ es ortogonal a f_1 y f_2, lo que se llama principio de ortogonalidad. Simbólicamente:

$$f_1^T \left(Y - F\hat{\Theta} \right) = 0 \tag{2.50}$$

$$f_2^T \left(Y - F\hat{\Theta} \right) = 0 \tag{2.51}$$

Estas dos condiciones se reducen a:

$$F^T = \left(Y - F\hat{\Theta} \right) = 0 \tag{2.52}$$

Que es exactamente la derivada de la ecuación normal de la ecuación (2.43), mediante la diferenciación directa.

2.3.6. Heurísticas de minimización del error

En este capitulo repasamos varias técnicas de optimización basadas en su capacidad de determinar la dirección de búsqueda de la derivada de una función objetivo, entre estas técnicas, citamos el método de steepest descent, y el método de Newton, ya que se puede considerar los algoritmos posteriores destinados a este fin como un compromiso de estas dos técnicas, en este sentido, describimos el método de Levenberg-Marquardt. Estos métodos.

2.3.6.1. Métodos de descenso

En este párrafo nos centramos en la minimización de funciones de error E de valores reales, definidas en un espacio de entradas *n*-dimensional $\Theta = \left[\theta_1, \theta_2, \ldots, \theta_n\right]^T$. Encontrando un posible mínimo local $\Theta = \hat{\Theta}$ que minimiza $E\left(\Theta\right)$ es primordial.

Generalmente una función objetivo E podría tener una forma no-lineal respeto al parámetro Θ. Debido a la complejidad de E, recurrimos a un algoritmo iterativo para explorar eficazmente el espacio de entrada. En los métodos de descenso iterativo, el punto siguiente Θ_{sig} es, partiendo del punto actual Θ_{act}, un paso hacia abajo siguiendo el vector dirección d:

$$\Theta_{sig} = \Theta_{act} + \eta d \tag{2.53}$$

Donde η es el paso con que se avanza en dicha dirección. En la literatura neuro-difusa, se refiere a η como coeficiente de aprendizaje, la formula (2.53) se puede escribir también:

$$\Theta_{k+1} = \Theta_k + \eta_k d_k \qquad \left(k = 1, 2, 3, \ldots\right) \tag{2.54}$$

Donde *k* es la iteración actual, y Θ_k trata de converger hacia un mínimo (local) Θ^*.

Los métodos de descenso iterativo calculan el *k*-ésimo paso $\eta_k d_k$ mediante dos procesos: determinando la dirección d, y a continuación calculando el *paso* η. Por lo que el punto siguiente Θ_{sig} ha de satisfacer la inecuación siguiente:

$$E\left(\Theta_{sig}\right) = E\left(\Theta_{act} + \eta d\right) < E\left(\Theta_{act}\right) \tag{2.55}$$

La dirección de descenso d se puede determinar basándose el gradiente g de una función objetivo $E : R^n \to R$, diferenciable en Θ:

$$g\left(\Theta\right)\left(= \nabla E\left(\Theta\right)\right) = \left[\frac{\partial E\left(\Theta\right)}{\theta_1}, \frac{\partial E\left(\Theta\right)}{\theta_2}, \ldots, \frac{\partial E\left(\Theta\right)}{\theta_n}\right]^T \tag{2.56}$$

Una clase de los métodos de descenso basados en el gradiente se escriben de forma que se puedan determinar las direcciones de descenso factibles desviando los gradientes por multiplicación por **G** (*gradientes desviados)*:

$$\Theta_{sig} = \Theta_{act} - \eta G g \tag{2.57}$$

Donde **G** es una matriz definida positiva.

El caso ideal seria encontrar un valor de Θ_{sig} que satisfaga:

$$g\left(\Theta_{sig}\right) = \left.\frac{\partial E\left(\Theta\right)}{\Theta}\right|_{\Theta=\Theta_{sig}} = 0 \tag{2.58}$$

Esta es una condición necesaria pero no suficiente.

En la práctica, sin embargo, es difícil resolver la ecuación (2.58) analíticamente. Para minimizar la función objetivo, se repiten los procesos de descenso hasta cumplir una de las condiciones de parada siguientes:

1. el valor de la función objetivo es lo bastante pequeño;
2. el tamaño del vector gradiente G es inferior a un valor especificado por nosotros;
3. superar el tiempo de computación especificado.

2.3.6.2. El método de steepest descent

El método del steepest descent, es uno de los métodos de minimización de funciones definidas en un espacio pluridimensional. A pesar de su lenta convergencia, es el método más usado en la optimización no-lineal gracias a su simplicidad.

Cuando $G = I$ (I es la matriz identidad), la ecuación (2.57) que se llamara formula de steepest descent sería:

$$\Theta_{sig} = \Theta_{act} - \eta g \tag{2.59}$$

Si se analiza el caso de una función objetivo cuadrática, el método de steepest descent genera solamente dos direcciones ortogonales determinadas por el punto de salida como muestra la figura 2.8.

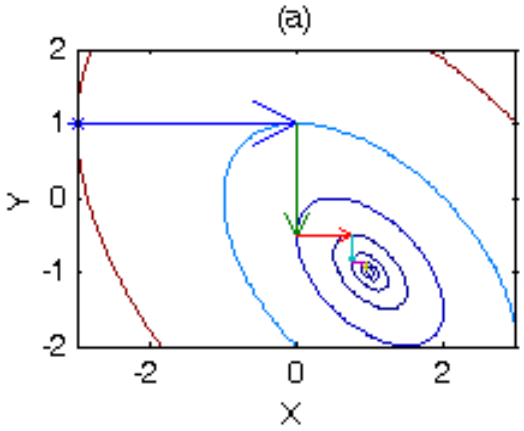


Figura 2.8: *El progreso del descenso para la minimización de la función cuadrática* $E(x, y) = x^2 + xy + y^2 - x + y$ *mediante el método de steepest descent.*

2.3.6.3. El método de Newton

Se puede determinar la dirección de descenso d usando las *segundas derivadas* de la función objetivo E. Para una función objetivo continua, si la posición inicial Θ_{act}

es lo bastante cerca de un mínimo local, se podría aproximar la función objetivo de forma cuadrática:

$$E(\Theta) \approx E(\Theta_{act}) + g^T(\Theta - \Theta_{act}) + \frac{1}{2}(\Theta - \Theta_{act})^T H(\Theta - \Theta_{act}) \quad (2.60)$$

Donde H, la matriz de Hess, es la *derivada segunda parcial* de $E(\Theta)$.

La ecuación anterior es la expansión en series de Taylor de $E(\Theta)$ elevada hasta el segundo orden. Se han omitido los términos de orden superior asumiendo que $\|\Theta - \Theta_{act}\|$ es suficientemente pequeña.

Puesto que la ecuación (2.60). es una función cuadrática de Θ, su punto mínimo $\hat{\Theta}$ se puede encontrar, diferenciando la ecuación (2.60). y poniéndola igual a 0. Lo que nos llevaría a un conjunto de ecuaciones lineales:

$$0 = g + H(\hat{\Theta} - \Theta_{act}) \quad (2.61)$$

Si H tiene inversa, tendríamos una solución única.
Por consiguiente, la formula de Newton de escribe de la manera siguiente:

$$\hat{\Theta} = \Theta_{act} - H^{-1}g \quad (2.62)$$

El término $H^{-1}g$ es llamado paso de Newton.

La formula general de la ecuación se reduce a la formula de Newton cuando $G = H^{-1}$ y $\eta = 1$.

La formula de Newton previamente descrita requiere, frecuentemente, adecuaciones antes de ser implementada. Si Θ_{act} está alejado del mínimo local Θ^*, la formula podría no producir un descenso en gradiente debido a los términos omitidos en la expansión en series de Taylor de la función $E(.)$.

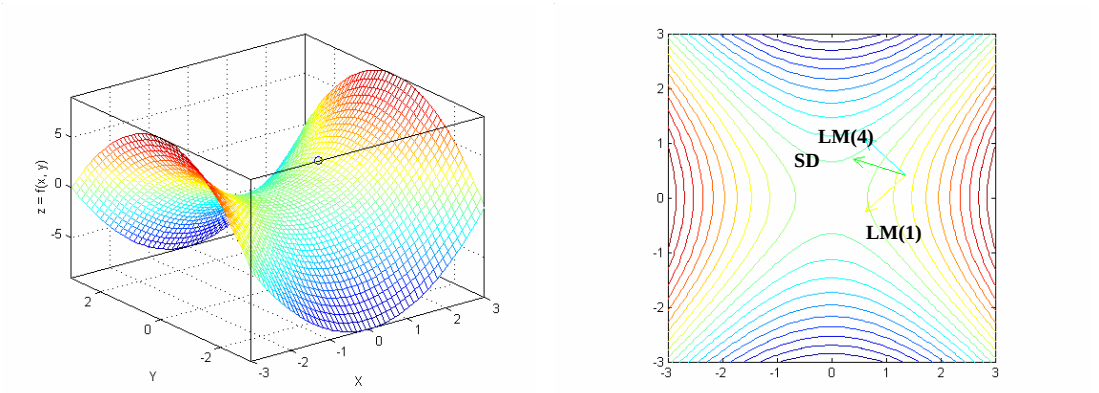
2.3.6.4. La formula de Levenberg-Marquardt

Si la matriz de Hess no es definida positiva, la dirección de Newton podría apuntar a hacia un máximo local. Se puede alterar la matriz de Hess H añadiéndole una matriz P para hacerla definida positiva, Levenberg [Levenberg 1944] y Marquardt han introducido esta noción en los problemas de los mínimos cuadrados. Más tarde [Goldfeld et al 1966] aplicó este concepto a la formula de Newton, cunado $P = \lambda I$, la ecuación (2.62) se escribiría:

$$\Theta_{sig} = \Theta_{act} - \left(H + \lambda I\right)^{-1} g \tag{2.63}$$

Las modificaciones de Levenberg-Marquardt hacen que la formula de Newton más robusta.

Dependiendo del valor de λ, el método transita suavemente entre 2 métodos: el método de newton $\left(\lambda \to 0\right)$ y el método del descenso en gradiente $\left(\lambda \to \infty\right)$.



(a) (b)

Figura 2.9: *El método de Levenberg-Marquardt: (a) Una superficie cuadrática* $E\left(\theta\right) = E\left(x,y\right) = x^2 - y^2$ *; (b) 2 direcciones de Levenberg-Marquardt: Lambda(1) y Lambda(4), y la dirección del descenso en gradiente. Las direcciones están normalizadas en estas figuras.*

La figura 2.9 ilustra una función hiperbólica paraboloide definida como $E(\theta) = E(x, y) = x^2 - y^2$.

2.4. Conclusión

Las redes neuronales basadas en funciones de base radial descritas en este capítulo constituirán la herramienta básica en la construcción del algoritmo propuesto en este trabajo, de manera hibrida con otras técnicas bio-inspiradas se realizarán un conjunto de algoritmos orientados al problema de la optimización y más concretamente a la tarea de la aproximación de funciones.

Capítulo 3/
Nuevas arquitecturas bio-inspiradas para la aproximación de funciones

En los capítulos anteriores, se han descritos varias metodologías de aprendizaje bio-inspirado. Estas técnicas materializadas en diferentes variantes de las redes neuronales artificiales (como es el MLP, el adaline descritos en el apartado 1.4) así como las heurísticas que intervienen en el desarrollo de un algoritmo basado en un red neuronal como las técnicas de inicialización de centros de clusters definidas en el apartado 2.4.2, y las técnicas de aprendizaje supervisado 2.4.6 e in-supervisado 1.4.4

En el este capítulo se describe un algoritmo inteligente compuesto esencialmente de una RBFN de salida única y adaptada al problema de aproximación de funciones. El algoritmo propuesto se caracteriza por una estructura que combina de manera integrada una red RBFN y un algoritmo genérico con el objetivo de construir un sistema inteligente híbrido y complementario.

A lo largo del diseño de este trabajo, y como se describe a continuación, tanto las diferentes técnicas que entran en la composición del propio algoritmo como los parámetros que influyen en su comportamiento, han sido introducidos siguiendo las directrices de fiabilidad y robustez que rigen las heurísticas bio-inspiradas.

3.1 Estructura bio-inspirada para la aproximación de funciones

El algoritmo S-RBF pretende crear generaciones de individuos y evaluarlas iterativamente y hasta alcanzar una condición de parada, de forma que en la ultima generación estarán presentes los individuos con mejor evaluación a lo largo del proceso, dicho proceso se encarga de determinar de manera exclusiva los parámetros de las RBFs que integran nuestra red neuronal, esta característica proporciona una gran flexibilidad y ahorro en términos computacionales, los operadores genéticos que entran en la selección y la evolución de los individuos son totalmente adecuados a la estructuración de las RBFs.

La figura 3.1 nos da una visión simplificada del desarrollo a seguir para la aplicación del algoritmo S-RBF.

El algoritmo genético trabaja con los parámetros inicializados previamente mediante el algoritmo de clustering especialmente diseñado para la tarea de aproximación de funciones, por lo que dichos parámetros ya se encuentran en un espacio de búsqueda reducido, permitiendo de este modo un ahorro considerable en tiempo de computación. El proceso iterativo del algoritmo genético permite la generación de una secuencia compuesta por los parámetros de la función de base radial que se persigue, dicho proceso está guiado por unos operadores genéticos para aproximación de funciones. En efecto, los operadores de cruce y mutación aplicados en este trabajo han sido, tras un proceso de simulaciones que se detallaran más adelante, adaptados para este tipo de problemas, además se han introducido cambios en la manera de selección de la población inicial para que sea diferente en cada repetición del algoritmo y así obtener un resultado más generalizado.

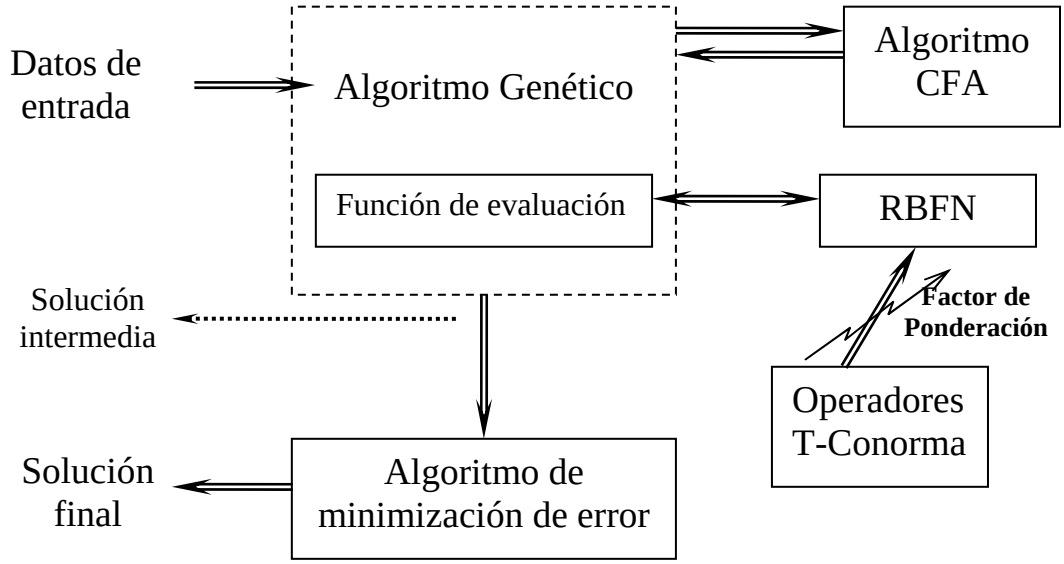


Figura 3.1: D*iagrama esquemático del algoritmo S-RBF, la red neuronal actúa de manera integrada dentro del algoritmo genético.*

Al igual que otros trabajos anteriores en el área de aproximación de funciones, el algoritmo SRBF introduce una fase de clustering de datos, en este caso se recurre al algoritmo CFA, que funciona de manera a incrementar el número de centros de clusters en aquellas áreas de la función objetivo que presenta más variabilidad de salida, en vez de hacerlo en las zonas de mayor número de variables de entrada, como viene siendo habitual en los algoritmos de clustering de datos convencionales. El algoritmo CFA participa de manera primordial en el comportamiento global de nuestra red, puesto que nos proporciona una idea muy concreta sobre el valor inicial de los parámetros que usar en el algoritmo genético.

Las funciones de base radial introducidas son de tipo Gaussianas, para las que hay que determinar un centro y un ancho, dichas funciones construyen la red neuronal de una sola salida tal y como se ha descrito en el apartado 2.3. Se ha trabajado exclusivamente con este tipo de funciones por su diseño, manejo y por la extensión que conocen en los trabajos relativos al problema de aproximación funcional.

El algoritmo SRBF introduce en su fase neuronal un concepto totalmente novedoso a la hora de combinar las RBFs, en efecto, las entradas de la capa oculta de nuestra red son a la vez tratadas mediante una combinación basada en operadores T-conorma de manera a convertir en ventaja el solapamiento entre funciones, y ponderadas mediante un peso que controla el aporte de cada neurona a la salida de la red. Dicho solapamiento está por supuesto controlado de manera a conseguir un equilibrio, evitando un sobre-entrenamiento totalmente negativo para el algoritmo en cuestión.

Une vez fijados los parámetros propios de las RBFs, y teniendo en cuenta que en una red neuronal viable, el número de vectores de entrada es superior al de neuronas que componen la capa oculta, los pesos de red asociados a cada función dentro del sistema de ecuaciones lineales de la fórmula (2.38) se calcularán de manera óptima aplicando un algoritmo de optimización local como pueden ser el método de los mínimos cuadrados ortogonales y la descomposición en valores singulares, que además sirve para clasificar las RBFs según su contribución a la reducción del error, y de este modo, eliminar aquellas funciones que menos aportan al conjunto, ganando en coste computacional sin perder en eficiencia.

Las estrategias anteriormente definidas hacen que dentro de la red los individuos estén en continuo proceso de entrenamiento y competencia entre si, esta combinación hace que la respuesta general sea aun mejor que la de un algoritmo que trata únicamente la selección y el entrenamiento de individuos.

La red final obtenida es un conjunto de algoritmos que, partiendo de un número de puntos pertenecientes a una función, nos proporcionan una aproximación a esta función con un margen de error al que se comparará con otros trabajos que tratan el mismo problema.

En las secciones que siguen a continuación, detallaremos los componentes básicos de nuestra red, el efecto que tienen los diferentes operadores aplicados en la construcción de nuestro sistema, así como la justificación de los valores escogidos, se detallarán simulaciones básicas comparativas, que han servido de manera preponderante en la realización de nuestra red.

3.2 Componentes elementales del algoritmo propuesto

El algoritmo que presenta este trabajo es el resultado de una combinación de varias técnicas bio-inspiradas alrededor de una de las aportaciones novedosas como es la función de evaluación del el algoritmo S-RBF, tanto los operadores genéticos como los componentes del algoritmo de clustering de datos han sido idealizados mirando por unos objetivos principales:

- Minimización del error: el objetivo final del algoritmo es aproximar una función de una o más dimensiones reduciendo el error, el valor del error medio cuadrático normalizado diera entonces el indicador absoluto de la fiabilidad del conjunto.
- Complejidad del sistema: se trata de construir un sistema robusto y eficaz, para ello, se han puesto a prueba cada una de las diferentes fases del desarrollo del algoritmo y los parámetros que la componen, el objetivo final es eliminar toda redundancia posible.
- El coste computacional: principalmente en las fases de computo de funciones RBF, y de la parada del algoritmo, con el objetivo de reducir la carga computacional en cada ejecución.

3.2.1 Representación esquemática del algoritmo propuesto

A partir de la descripción anterior, se puede representar en formato de organigrama las diferentes fases de nuestro algoritmo.

La inicialización de nuestra red se hace mediante un algoritmo de clustering de datos, que a partir de los datos de entrada pertenecientes a la función que se quiere aproximar, propone un emplazamiento inicial de los centros de funciones RBF, el objetivo es entrar en la fase del algoritmo genético con una idea previa sobre la disposición de las funciones RBF dentro del espacio de búsqueda. De esta manera se evita que el algoritmo genético lance una búsqueda ciega de los centros de RBFs, y se aprovecha de esta manera del algoritmo de clustering CFA como medio fiable de agrupamiento de

datos. La introducción del algoritmo de clustering aumenta en parte la complejidad del sistema pero en cambio ayuda a reducir el coste computacional.

1. Cargar los conjuntos de datos de entrenamiento y test.
2. Inicializar los parámetros de las RBFs mediante Clustering.
3. Inicializar el algoritmo genético.
4. Aplicar operadores genéticos.
5. Aplicar el algoritmo S-RBF para evaluar el fitness.
6. Entrenar RBFs aplicando un el algoritmo de mínimos cuadrados.
7. repetir el algoritmo hasta el cumplimiento del criterio de parada.
8. Aplicar heurística de minimización del error.

Algoritmo 3.1: *Proceso de desarrollo del algoritmo propuesto.*

Una vez obtenidos los centros de clusters, el algoritmo genético, y mediante la aplicación de los operadores que lo componen, realiza una búsqueda iterativa que tiene como objetivo la optimización de los parámetros, dispuestos en cadena, de las RBFs (centro, ancho, y parámetro de ponderación).

3.2.2 Definición de los operadores T-conorma

El componente principal de un algoritmo genético es su función de evaluación, se encarga de cuantificar una solución de la manera más óptima, la función de evaluación que se aplica en este trabajo recoge cada cadena que representa una posible solución y la recompone como conjunto de parámetros para cada RBF de nuestra red, asimismo, se introduce una de las aportaciones de nuestro algoritmo y que se basa en aprovechar las propiedades T-conorma de las RBFs.

Definición 3.1: un operador T-conorma, en su formula generalizada $S(.,.)$, es la unión de dos funciones, y cumple las siguientes propiedades:

Delimitación	$S(1,1)=1,\ S(0,f)=S(f,0)=f$	
Monotonicidad	$S(f_1,f_2)\le S(f_3,f_4)$ si $f_1\le f_3$ y $f_2\le f_4$	(3.1)
Conmutatividad	$S(f_1,f_2)=S(f_2,f_1)$	
Asociatividad	$S(f_1,S(f_2,f_3))=S(S(f_1,f_2),f_3)$	

El operador T-conorma se puede exprimí de varias maneras, entre ellas distinguimos:

Máximo:	$S(f_1,f_2)=\max(f_1,f_2)=f_1\vee f_2$	
Suma algebraica:	$S(f_1,f_2)=f_1+f_2-f_1.f_2$	(3.2)
Suma delimitada:	$S(f_1,f_2)=1\wedge(f_1+f_2)$	

La figura 3.2 representa una combinación por T-conorma de dos funciones gaussianas

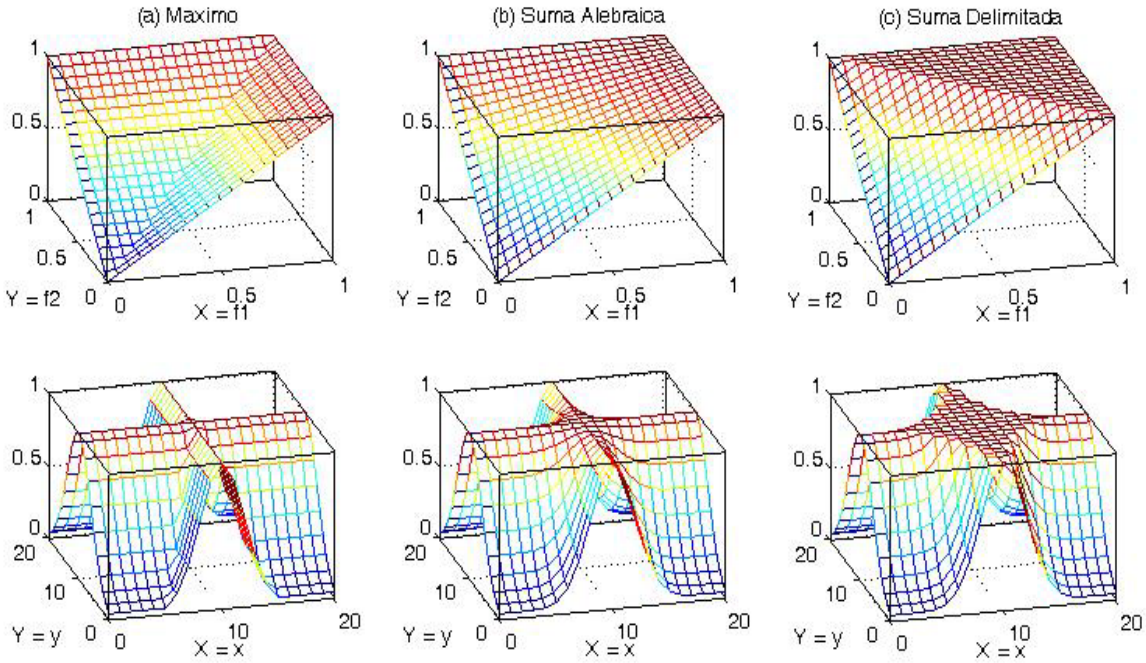


Figura 3.2: *Primera línea: Tres operadores de T-conorma:* $S_{\max}(f_1,f_2)$, $S_{sa}(f_1,f_2)$, *y* $S_{sd}(f_1,f_2)$*; Segunda línea: las superficies correspondientes para* $f_1=gaussiana(x,[10,2])$ *y* $f_2=gaussiana(y,[10,4])$.

La interpretación y la aplicación de este concepto en una red neuronal dirigida a la tarea de la aproximación de funciones es un enfoque totalmente novedoso y que implica un diseño diferente al de una red RBF convencional, puesto que algunas características de estas redes, como puede ser el solapamiento entre funciones o el peso asignado a cada una de ellas, tienen un impacto diferente sobre el resultado final.

Una vez establecidos los parámetros propios de cada RBF, se realiza un entrenamiento de los pesos de cada función dentro de la red neuronal, en esta fase aplicamos un algoritmo LMS, debido a su simplicidad y al coste computacional reducido que supone. La aplicación del algoritmo LMS, en un proceso reducido en número de iteraciones, proporciona una idea aproximativa sobre el peso de red de cada RBF.

El algoritmo propuesto en este trabajo, y teniendo en cuenta el contexto coevolutivo en el que se desarrolla, asigna un crédito característico a cada función según su aportación a la red, este parámetro, llamado, de ponderación es diseñado como parámetro propio de la RBF y va evolucionando a lo largo del proceso iterativo del algoritmo genético, este planteamiento supone una novedosa aportación en el campo de la redes neuronales artificiales y su implementación proporciona unos datos muy interesantes a lo largo de la evolución del proceso iterativo. Este parámetro, arranca con un valor neutro para todas las funciones y, a lo largo del proceso de entrenamiento, evoluciona según si la función que representa abarca un número de puntos de entrada, su grado de solapamiento a las funciones contiguas.

En la última fase del proceso, se aplica un algoritmo de Levenberg-Marquardt, con el objetivo de ajustar definitivamente los parámetros de la red, obteniendo una configuración final optimizada.

3.2.3 El conjunto de datos de entrada

Como en los algoritmos basados en RBFN convencionales, el algoritmo S-RBF precisa un conjunto de entrenamiento del que se extrae información sobre los valores de los parámetros y al que, posteriormente, los operadores genéticos recurren para la determinación de la estructura definitiva de la red. En paralelo a este proceso, se trabaja

con un conjunto llamado test; este conjunto tiene una estructura diferente, con una componente aleatoria, y permite valorar la capacidad de generalización de la red.
A lo largo del proceso, el conjunto de datos de entrada es único, tiene un tamaño constante, y se mantiene en cada generación la presencia del mejor individuo. En la parte genética del algoritmo S-RBF los individuos, cuyo número es relativo al número de neuronas llevan la información relativa a los parámetros que caracterizan la función RBF que integra dicha neurona.

3.2.4 Codificación de los individuos

Los algoritmos genéticos convencionales admiten para su proceso iterativo cadenas binarias a las que se aplican los operadores genéticos. En nuestro caso, donde el proceso genético trata de aportar soluciones a una red neuronal, los individuos constituyen parámetros básicos para construir RBFs, esta complejidad obliga ver el tema de la codificación de manera distinta, una de las opciones que se barajan es codificar los individuos de valores “flotantes” en cadenas binarias para procesos posteriores, pero este razonamiento trae unas complicaciones que acompañan los procesos de codificación y descodificación. Además, y puesto que en este trabajo se perfeccionan nuevos operadores genéticos dirigidos al objetivo del optimización de funciones, del mismo modo se les adapta al trato de individuos más complejos como se describirá a continuación.

3.3 Creación de la población inicial

Usando un algoritmo genético como herramienta de entrenamiento por iteraciones, y tanto en los problemas de aproximación de funciones como en los de clasificación, la solución final difiere bastante de las que se generan de manera aleatoria en la población inicial, lo que daría a entender que toda solución final es el resultado exclusivo tanto del proceso iterativo como de la influencia de los diferentes operadores genéticos y por lo tanto completamente independiente de la población inicial, sin embargo, se ha

comprobado que, partiendo de una población inicial adecuada, se logra disminuir de manera consecuente el tiempo de ejecución del algoritmo en cuestión.

La opción de partir desde una inicialización básicamente aleatoria, tiene como ventaja su sencilla implementación, así como su reducido coste computacional ya que se basa en la generación de una población aleatoria. En cambio, el hecho de establecer una determinada situación inicial, reduce el espacio de búsqueda manteniendo una población compuesta por soluciones potenciales, esta operación es aplicable siempre y cuando la heurística utilizada para una inicialización previa sea fiable ya que este punto en concreto del algoritmo es de sentido único, es decir, la información generada por un mecanismo, como en nuestro caso será un algoritmo de clustering de datos, es definitiva y no tiene posibilidad de corrección. Los demás inconvenientes que puede suponer tal implementación, como el aumento de la complejidad del sistema son perfectamente asumibles.

3.3.1 Inicialización de los centros

Basándose en lo descrito en capítulos anteriores, en este trabajo se recurre a un algoritmo especialmente diseñado para la tarea de aproximación de funciones como es el algoritmo CFA, que partiendo de un análisis del conjunto de datos de entrenamiento proporcionan una partición definitiva de este conjunto en grupos, y además, tiene la capacidad de redistribuir los centros de clusters según su aporte al sistema.

El algoritmo de clustering, aplica unas pautas referentes a la distancia que separa cada centro de cluster a los de vectores de entrada que lo componen, realiza una distribución según la ecuación:

$$J = \frac{\sum_{i=1}^{m} \sum_{k, x_k \in C_i} \left\| x_k - c_i \right\|^2}{m} \tag{3.3}$$

Es la ecuación (2.24) teniendo en cuenta que en la primera inicialización, el parámetro ω_{ki} adquiere un valor neutro puesto que en este punto del proceso, todavía no se tiene una idea de la influencia de cada RBF en particular sobre el conjunto de la red.
Tras este paso, y en un proceso iterativo, el algoritmo CFA continúa realizando las actualizaciones y migraciones pertinentes hasta alcanzar la condición de parada previamente definida.

3.3.2 Inicialización de los anchos

Acompañando el algoritmo del clustering de datos, se aplica una heurística de determinación de anchos de RBFs, para esta tarea, se recurre a las técnicas de los *k* vecinos más cercanos y de la distancia media de los vectores de entrada descritas en el apartado 2.4.4 del capítulo anterior. La mayor diferencia a la hora de implantarlas en un algoritmo basado en RBFNs es el grado de solapamiento menor que presenta la de los vectores de entrada más cercanos. Si bien, se obtienen diferentes resultados aplicando una u otra técnica, no se puede afirmar de manera categórica que una es mejor que otra.
Una inicialización de los anchos de RBFs aplicando la técnica de los vectores de entrada más cercanos cumple la siguiente ecuación:

$$\sigma_i = \frac{\sum_{k, x_k \in C_i} \left\| x_k - c_i \right\|}{k} \tag{3.4}$$

Como se nota, su valor depende matemáticamente hablando de la distancia entre el centro de la función en cuestión y cada uno de los puntos que abarca.

3.3.3 Inicialización de los pesos de la red

Como ocurre a la hora de inicializar los centros de las RBFs, cuando hay que establecer un valor matemático de la influencia de cada RBF sobre el conjunto de la red, en el caso

de los pesos de la red, se inicializa con un valor igual para todas las funciones, en su momento, el algoritmo LMS, seria el encargado de optimizar los valores de estos pesos.

3.3.4 Inicialización del parámetro de ponderación

El parámetro de ponderación es una aportación novedosa, en el campo de las redes neuronales artificiales, durante el proceso genético, debido precisamente a que tiene como tarea la ponderación del efecto de los vectores de entrada en el posicionamiento de las RBFs, a este parámetro se le aplicara una inicialización aleatoria dentro de un intervalo de valores [-1,1], el proceso iterativo se encarga a continuación de asignar un valor a cada vinculo entrada-neurona según el trabajo que realiza ésta en la aproximación de dicho vector de entrada

3.4 El proceso genérico

Una vez inicializados los parámetros propios de las RBFs, estos son recompuestos en una población valida para un proceso basado en un algoritmo genético.

3.4.1 El tamaño de la población

El tamaño de la población tiene un efecto trascendental sobre el comportamiento de una red basada en un algoritmo genético, con una población de gran tamaño, se gana en diversidad y generalización aunque el coste computacional puede ser muy alto. Reduciendo el tamaño de la población, produce una mayor celeridad de procesamiento pero resulta al mismo tiempo una perdida de diversidad que puede ser nefasta para el desarrollo del proceso.

No existe en la bibliografía un método genérico para el establecimiento del tamaño de la población para un problema dado, ya que no pueden tener en cuenta ni las características particulares de cada red como la que realizamos en este caso y ni el problema tratado en si.

El análisis matemático de [Goldberg 1989-bis] tiene una sencilla conclusión sobre la implementación paralela de los algoritmos genéticos: escoger el tamaño de la población más grande posible, esto resulta inaplicable en arquitecturas cuyas características exigen un aumento en exponencial del tamaño de la red para obtener una optimización de la red.

En [Alander 1992], se estudia el efecto del aumento del tamaño de la población en un algoritmo genético en función de la complejidad del sistema, y se recurre a la codificación del problema en cadenas de bits.

Para nuestro trabajo, se ha realizado un estudio empírico del efecto del tamaño de la población sobre el resultado final y el tiempo de computación para así sacar una idea más completa sobre el efecto de tal cambio sobre la estructura que esta en construcción en esta memoria.

Aplicando el algoritmo S-RBF sobre unas funciones de diferentes características, se puede concluir que el tiempo de ejecución aumenta linealmente con el aumento del tamaño de la población, por otro lado, analizando el efecto del tamaño de la población sobre el error absoluto de aproximación de la red, se nota que hasta un cierto valor de este tamaño el efecto es positivo sobre la evolución del error, pero si se trabaja con valores superiores, el efecto puede ser nulo o incluso negativo, este comportamiento se podría explicar por el hecho de que al tratar con una población demasiado grande, el algoritmo no llega a procesar debidamente todas la soluciones posibles y la aplicación de los operadores genéticos no resulta concluyente.

Como resultado de lo descrito anteriormente, y teniendo en cuenta que el algoritmo genético arranca como paso posterior a unas heurísticas que realizan una inicialización selectiva de los componentes de la población, se ha decidido para determinar el tamaño de la población, aplicar el algoritmo S-RBF sobre las funciones $sintetica\_1d\_6n$ y $sintetica\_2d\_6n$, descritas en el capítulo 4, con valores ascendentes del tamaño de la población.

Estas dos funciones objetivo se les han realizado diez ejecuciones, usando un tamaño de la población cada vez más alto. Las tablas 3.1 y 3.2 representan los resultados obtenidos para la medida del error así como el tiempo medio de ejecución para cada caso.

Tamaño de la población	*Inicialización Aleatoria*			*Inicialización Selectiva*		
	Duración	*Medio*	*Desv.*	*Duración*	*Medio*	*Desv.*
10	100	0.1757	0.0144	238	0.1302	0.0080
20	129	0.1481	0.0154	289	0.1235	0.0112
30	168	0.1322	0.0169	350	0.1086	0.0097
40	182	0.1294	0.0163	411	0.0848	0.0054
50	267	0.1278	0.0148	462	0.0821	0.0065
100	688	0.1253	0.0159	932	0.0873	0.0120
150	770	0.1250	0.0163	1169	0.0810	0.0087
200	1095	0.1265	0.0182	1422	0.0815	0.0085
500	2467	0.1247	0.0140	3251	0.0822	0.0098
1000	4572	0.1232	0.0171	5163	0.0807	0.0051

Tabla 3.1: *Aplicación de valores ascendentes del tamaño de población a la función* $sintetica\_2d\_6n$ *a partir de una inicialización aleatoria de los vectores de la población inicial y otra mediante una inicialización selectiva.*

Relevamos de las tablas que en el caso concreto de la inclusión de las heurísticas para la inicialización de los individuos de la población, el aumento hasta un cierto valor del tamaño de la población provoca que el algoritmo mejore sus prestaciones, pero que se puede fijar éste entre 40 y 50 individuos presentando un grado alto de optimización, a partir de estos valores óptimos del tamaño de la población, el tiempo de ejecución aumenta considerablemente, mientras el error cuadrático medio no presenta una mejoría significativa.

3.4.2 El proceso de selección de individuos

Uno de los operadores genéticos que entran en el proceso iterativo de optimización de una solución es la selección de individuos.

Tamaño de la población	*Inicialización Aleatoria*			*Inicialización Selectiva*		
	Duración	*Medio*	*Desv.*	*Duración*	*Medio*	*Desv.*
10	29	0.0322	0.0032	75	0.0192	0.0035
20	38	0.0245	0.0022	92	0.0136	0.0014
30	52	0.0218	0.0029	113	0.0111	0.0020
40	57	0.0189	0.0042	134	0.0108	0.0023
50	84	0.0177	0.0018	149	0.0100	0.0019
100	225	0.0152	0.0035	306	0.0094	0.0014
150	253	0.0175	0.0058	382	0.0091	0.0021
200	360	0.0173	0.0040	475	0.0087	0.0006
500	818	0.0167	0.0031	1069	0.0086	0.0010
1000	1521	0.0169	0.0025	1704	0.0088	0.0013

Tabla 3.2: *Aplicación de valores ascendentes del tamaño de población a la función* $sintetica\_1d\_6n$ *a partir de una inicialización aleatoria de los vectores de la población inicial y otra mediante una inicialización selectiva.*

En este trabajo se han considerado varias funciones de selección, que tienen en común la aplicación del principio de selección natural de los mejores individuos para la siguiente generación, además, se han valorado las capacidades que tienen para abarcar todo el espacio de búsqueda, y sobre todo la cualidad de mantener un alto nivel de selección hasta la ultima generación, el caso del método de selección por ruleta es un ejemplo de la perdida de la intensidad de selección en cuanto la población converge hacia una solución.

Se ha optado por un método de selección basado en la distribución geométrica normalizada [Houck et al 1995], las soluciones se proyectan en un conjunto ordenado, y

se le asigna a cada una un valor P_i. La ordenación geométrica normalizada define el valor de P_i para cada individuo como:

$$P_i = \frac{q.(1-q)^{r-1}}{1-(1-q)^P} \tag{3.5}$$

Donde:

q es la probabilidad de seleccionar el mejor individuo,

r es el rango del individuo, con 1 como mejor rango,

P es el tamaño de la población,

Este método, permite escoger la mejor solución y mantenerla de una generación para la siguiente, además de tener la aleatoriedad necesaria para una tarea de prospección dentro de todo el espacio de búsqueda.

3.4.3 El operador de cruce

El operador de cruce tiene una especial relevancia en las redes basadas en RBFNs, ya que cada neurona tiene un efecto local es decir, responde a una zona específica del espacio de entrada y por consiguiente, el conjunto de neuronas se alimenta del aprendizaje competitivo que se ve favorecido por el efecto del cruce entre individuos.

Como se ha descrito anteriormente, el resultado de las heurísticas de inicialización son recompuestos en una sola cadena de solución a la que se aplican los siguientes operadores genéticos, incluido el cruce. Esto implica realizar una prevención importante, ya que no se puede permitir que se rompa la cadena en medio de un bloque de centros de RBFs, provocando la total perdida de la solución. Por lo tanto, los operadores de tipo cruce binario quedan totalmente descartados.

El propósito final de la aplicación del cruce en nuestro algoritmo es más que realizar cruces entre individuos, intercambiar información completa sobre neuronas. Al

finalizar, la red contendrá nueva información conservando el mismo número de neuronas que tenia al principio.

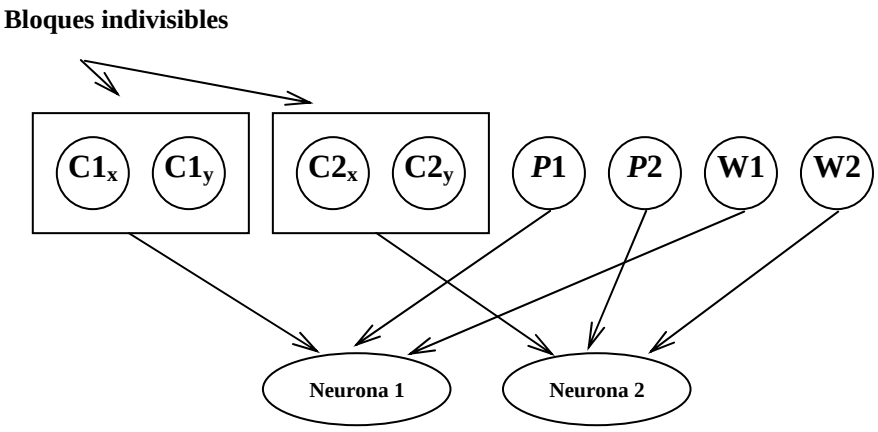


Figura 3.3: *Representación simplificada de una cadena compuesta de los parámetros de una RBFN.*

En nuestra estructura, implantamos un algoritmo de cruce aritmético, con restricciones. A lo largo del proceso genético, las operaciones de cruce solo serán permitidas en unos puntos concretos para no dividir la información que se mueve en conjunto como la relacionada a los centros de RBFs y crear soluciones totalmente divergentes.

3.4.4 El operador de mutación

El objetivo principal que se persigue aplicando el operador de mutación en nuestro algoritmo es lograr una exploración global y aleatoria del espacio de búsqueda, y así, reducir la dependencia de la población inicial y salir de los posibles óptimos locales en los que el algoritmo genético pueda caer.

En una operación de mutación básica, en la que se realizan cambios aleatorios en un individuo de la población, todos los individuos tienen la misma probabilidad de ser escogidos, y la alteración afecta una zona del individuo totalmente aleatoria [Holland 1975].

Del mismo modo que en la aplicación de cruce, el operador de mutación debe adaptarse a las características de la red que construimos.

La mutación que aplicamos en esta memoria, realiza una alteración del valor del parámetro del centro, del ancho de una RBF o del parámetro de ponderación, quedando excluido el peso, aunque forma parte de la cadena de solución puesto que de esta tarea de encargaría de manera más adecuada una heurística de optimización de pesos. Además, al no tratarse de una cadena con codificación binaria, sino de un conjunto de valores que representan los parámetros de unas RBFs, toda substitución debe realizarse respetando los rangos de valores que pueda tomar el parámetro en cuestión.

Habiendo descartado los parámetros del peso del proceso de mutación, notar que, también, entre los parámetros de los centros y de los anchos de RBFs existen ciertas características propias que los diferencian. Efectivamente, en el caso de los centros, hay que reflejar en el operador que se construye, el hecho de que las necesidades cambian según se desarrolla el proceso iterativo del algoritmo genético, y se va aproximando a una solución optima. Si en el principio del proceso, la prioridad es de abarcar la totalidad del espacio de búsqueda mediante alteraciones aleatorias, y de este modo no caer en una convergencia rápida y mala, a medida que se va mejorando la solución, se hace más apropiado realizar cambios locales en los centros, es decir, cambiar el valor del centro por uno vecino, haciendo que la RBF se mueva en un espacio reducido, y afinando así la optimización.

En [Houck et al 1995], se introducen varios modelos de mutación, de los que destacamos [unifMutation] y [adjswapMutation] que han sido adaptados para nuestro algoritmo.

- unifMutation reemplaza el valor de un parámetro de la cadena por otro escogido aleatoriamente entre los posibles valores que pueda tomar este parámetro, el resultado, es un individuo *hijo* que es idéntico al padre excepto en el punto de mutación donde la alteración responde a la siguiente ecuación:

$$hijo\left(P\_mut\right) = \min_{padre} + red\left(a \cdot rango_{padre}\right) \tag{3.6}$$

Donde *a* es un número aleatorio en 0 y 1, y *red* es una función de redondeo hacia el valor más cercano entre los que componen el espacio de búsqueda.

- adjswapMutation realiza un trueque entre dos genes adyacentes. el individuo generado es idéntico al padre excepto en dos posiciones que se define de la menar siguiente:

$$\begin{aligned} P\_mut &= red\left(a \cdot \left(rango_{padre} - 1\right) + 0.5\right) \\ hijo\left[P\_mut; P\_mut + 1\right] &= padre\left[P\_mut + 1; P\_mut\right] \end{aligned} \tag{3.7}$$

3.4.5 Estimación de los valores opcionales del cruce y de la mutación

Con el objetivo de verificar el valor idóneo para las tasas de aplicación del cruce y la mutación, se han realizado varias ejecuciones en condiciones idénticas, variando en cada vez los valores de estos parámetros. Como en el caso de la determinación del tamaño de la población se han realizado 10 ejecuciones para cada valor de los que se resaltan, el mejor, el peor, la media y la desviación estándar, Los valores que pueda tomar cada tasa han sido escogidos de manera escalonada. El objetivo final es poner los operadores genéticos en las condiciones ideales para el funcionamiento del conjunto del algoritmo.

Los resultados obtenidos son presentados en las tablas siguientes:

Tamaño de la población	*Inicialización Aleatoria*				*Inicialización Selectiva*			
	Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
0	0.0209	0.0255	0.0226	0.0059	0.0181	0.0208	0.0186	0.0033
0,5	0.0191	0.0221	0.0205	0.0042	0.0167	0.0179	0.0172	0.0016
1	0.0170	0.0208	0.0188	0.0049	0.0162	0.0177	0.0167	0.0018
2	0.0173	0.0195	0.0181	0.0032	0.0164	0.0180	0.0169	0.0028
4	0.0210	0.0248	0.0219	0.0033	0.0180	0.0197	0.0188	0.022

Tabla 3.3: *Estudio del efecto de la probabilidad de aplicación del cruce a la función "sintética 1_12n".*

Tamaño de la población	*Inicialización Aleatoria*				*Inicialización Selectiva*			
	Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
0	0.0136	0.0149	0.0145	0.0022	0.0082	0.0095	0.0085	0.0011
0,5	0.0108	0.0122	0.0115	0.0008	0.0060	0.0074	0.0065	0.0015
1	0.0115	0.0126	0.0119	0.0017	0.0062	0.0075	0.0069	0.0009
2	0.0117	0.0131	0.0125	0.0020	0.0071	0.0080	0.0074	0.0009
4	0.0120	0.0139	0.0128	0.0015	0.0069	0.0089	0.0077	0.0012

Tabla 3.3: *Estudio del efecto de la probabilidad de aplicación del operador de mutación sobre la función "sintética 1_12n".*

Analizando los resultados anteriores, se puede decir que las diferencias no son tan pronunciadas como para considerar incluir más valores intermedios, pero sí suficientes para justificar este experimento y concluir que:

- en lo que se refiere al cruce: se nota que con un valor medio, el conjunto se comporta mejor, por lo que se considera mejor probabilidad de aplicación del cruce entre 1 y 2.
- En el caso de la mutación, se nota que un valor medio bajo entre 0.5 y 1, genera resultados más óptimos.

Para concluir, recordamos que la variación de los parámetros estudiados anteriormente, no tienen un efecto concluyente sobre el tiempo de ejecución del algoritmo, ni en general sobre el coste computacional o la complejidad del sistema.

3.5 La función de evaluación

En los problemas de optimización convencionales basados en algoritmos evolutivos, la función de evaluación mide la aptitud de cada solución permitiendo al algoritmo genético, seleccionar las soluciones que merezcan pasar a la generación siguiente. En casos frecuentes [Whitehead 1996], esta evaluación no pasa de una simple ordenación aritmética de las soluciones.

La función de evaluación propia al algoritmo genético, es considerada en este trabajo, una fase completa en si. El valor del fitness es en este caso asignado a toda una red de neuronas que representa una posible solución. La aptitud de una solución, en un problema de aproximación de funciones basado en RBFNs se mide calculando su capacidad de aproximar aquellos puntos de la función que no forman parte del conjunto de vectores de entrenamiento.

En el trabajo actual se remplaza la asignación de aptitud a un individuo relativamente al trabajo que aporta para la consecución de la solución final por calcular la aptitud de toda la red en su conjunto, este concepto además parece más razonable del punto de vista neuronal en el que el objetivo es aproximar una función mediante la simbiosis, en una red neuronal, de las RBFs cuyos parámetros van siendo alterados a lo largo del proceso.

El hecho de mantener los parámetros de las cadenas de individuos en forma de valores flotantes, y no someterlas a una codificación binaria por ejemplo, supone una complicación extra, y de hecho este tipo de codificación es novedoso y de reciente implantación. Esta elección hace que sea más complicado ordenar los individuos de la población, sin embrago, tiene la gran ventaja de exponer de una manera mucho más explicita las características de cada individuo facilitando la valoración de los mismos. Tras valorar ambas posibilidades, se concluye que la segunda configuración es mucho más beneficiosa para el comportamiento del algoritmo en su conjunto, y justifica plenamente el aumento en complejidad que supone.

3.5.1 Recomposición de la solución

Se ha descrito en los apartados anteriores como, tras la inicialización de los centros de las RBFs mediante un algoritmo de clustering de datos, y de los anchos mediante la técnica de los vecinos más cercanos, estos parámetros propios se recomponen en una cadena de individuos para formar la población inicial. La función de evaluación de nuestro algoritmo genético recoge estas cadenas de individuos y las recombina en forma de una red en la que cada neurona de la capa oculta lleva la información completa de la función base que la compone.

3.5.2 Aplicación del operador T-conorma

En este paso se requiere una atención especial ya que los datos recogidos tienen que ser adecuadamente recompuestos, teniendo en cuenta que el algoritmo que está en construcción va a ser totalmente independiente del rango de los datos de entrenamiento de la función objetivo así como del espacio en dimensiones en el que se encuentra.
La construcción de la red neuronal, se realiza en este trabajo de una manera totalmente distinta a lo que se ha hecho hasta el momento en el campo de las redes neuronales basadas en RBFs y supone una de las mayores aportaciones de esta memoria.

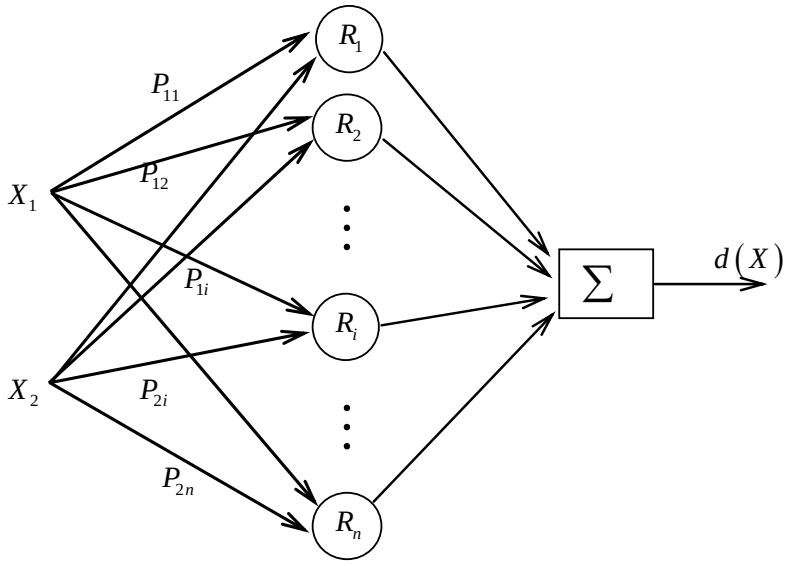


Figura 3.4: R*epresentación simplificada de una red de 2 entradas y n neuronas, el factor de ponderación influye en los vínculos entre vectores de entrada y neuronas.*

A partir de la representación simplificada de una red neuronal de varias entradas, una capa oculta integrada por RBFs Gaussianas y una única salida como muestra la figura 3.4, representamos la entrada a la capa oculta de la manera siguiente:

$$R_i = \sum_{j=1}^{2} P_{ji} \exp\left[-\frac{\left(X_j - u_i \right)^2}{2\sigma_i^2} \right] \tag{3.8}$$

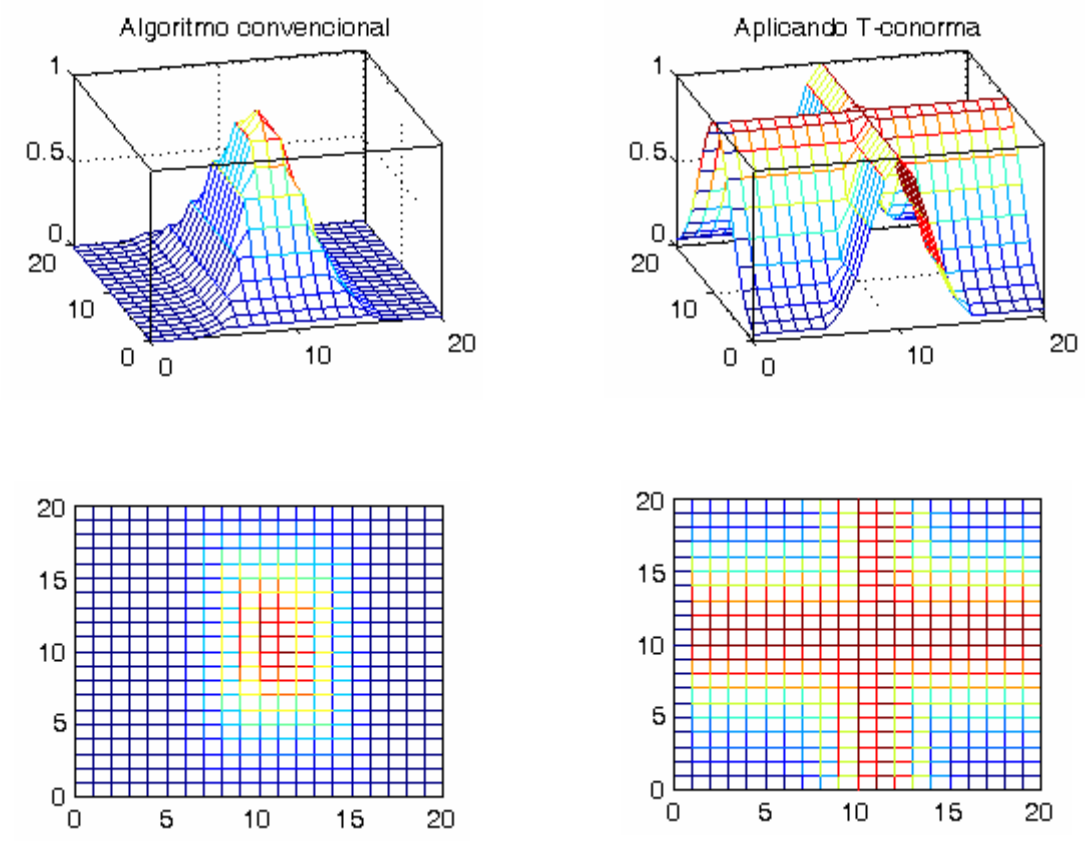


Figura 3.5: *Función RBF Gaussiana; (a) representación convencional;(b) Efecto de la implantación del operador T-conorma.*

En este trabajo, interpretamos la entrada a la neurona *i* de la capa oculta como la unión de los datos de entrada a esta misma, la idea es, respetando unas directrices de diseño que se detallaran más adelante, que se aproveche el espacio ocupado por cada una de la funciones RBF propias a cada entrada para abarcar de manera diferente los punto de datos de entrada. Además, para completar la estrategia, introducimos un nuevo parámetro propio a cada vínculo entrada-neurona, que se encarga de ponderar la unión entre capa de entrada y capa oculta, y representar el aporte de cada RBF para cada vector de entrada en particular. Este parámetro se inicializa de forma aleatoria dentro de un rango de valores predeterminado para que posteriormente, y mediante el proceso

evolutivo, se vaya afinando hasta obtener una configuración la más optimizada posible de la participación de cada neurona para abarcar cada vector de entrada.

Esta estrategia parte del hecho de que en un algoritmo convencional, cada RBF está sometida a la influencia "restadora" de las entradas y que solo se utilice el espacio de coincidencia entre las entradas, es decir la intersección entre las curvas como muestra la figura 3.5(a). Este comportamiento, si se invierte esta idea, como muestra la figura 3.5(b), haría que cada RBF aumente el espacio que ocupa dependiendo de los puntos de entrada que abarca. El valor del parámetro *P* tiene un efecto primordial en la ponderación entre los elementos de las capas de entrada y oculta, a medida que crece significa que la RBF aproxima de manera directa el vector de entrada, y en el otro sentido, un valor mínimo significa que la RBF no tiene ningún aporte a la aproximación de este punto de dato. Por lo tanto el proceso evolutivo aplicado al parámetro *P* completa la nueva estrategia que se aplica en esta fase del algoritmo.

3.5.3 Aproximación de los pesos de la red

Se han descrito en los capítulos anteriores diferentes métodos para determinar los pesos de la red. Entre los más destacados, y de fácil implementación en el campo de las RBFNs están los que aplican el método de descenso en gradiente, como son el algoritmo de los mínimos cuadrados, también llamada la regla delta [Widrow & Hoff 1960] y la descomposición en valores singulares, ésta ultima tiene características diferentes que la hace mas conveniente de implantar en un contexto diferente al del método de los mínimos cuadrados.

3.5.3.1. Implantación del algoritmo LMS

El funcionamiento del método LMS consiste en ir calculando en cada iteración la diferencia al cuadrado entre la salida deseada y la salida propuesta por el algoritmo iterativo mediante la constitución de la red neuronal a partir de los individuos de la población.

Existen otras técnicas de optimización de los pesos de la red basadas en la resolución de sistemas de ecuaciones o simplemente matrices. Estas técnicas se pueden perfectamente aplicar en el campo de las RBFNs, ya que la formula general (3.9) se puede considerar como un sistema de ecuaciones lineales.

$$\begin{pmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n} \\ \vdots & \cdots & \cdots & \vdots \\ \phi_{n1} & \phi_{n2} & \cdots & \phi_{nn} \end{pmatrix} . \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{3.9}$$

La determinación de los pesos en este caso seria resolviendo este sistema.

Para ello, existen entre otras, la descomposición en valores singulares, y la descomposición de Cholesky.

El algoritmo LMS destaca por su poca complejidad puesto que en dos pasos:

- Calcular la diferencia entre la salida deseada y la salida obtenida $\sum_{i=1}^{m} \left(y_i - f_i^T w_i \right)^2$.
- Aplicar una variación según la formula (3.10)

$$w_{k+1} = w_k + \eta . y_k \tag{3.10}$$

Donde η es el coeficiente de aprendizaje, su valor determina el rango de variación de una iteración a otra. Aumentado el valor de η se pasa de una lenta aproximación, hasta provocar una oscilación divergente del peso.

Por otro lado, las técnicas basadas en la descomposición de matrices, aportan soluciones de gran exactitud, sin embargo, precisan el cumplimiento de unas condiciones previas para poder aplicarlas, tales como la concordancia de las dimensiones o la no-singularidad.

Este trabajo realiza la aproximación de una función objetivo mediante la construcción de una RBFN. La población a la que se aplica el algoritmo evolutivo esta constituida por los parámetros de las funciones RBF. En [Whitehead 1996], se trata de aproximar una RBFN mediante un método cooperativo comparativo basado en un algoritmo genético, y se aplica una combinación del algoritmo de mínimos cuadrados y la descomposición en valores singulares, esta estrategia tiene un razonamiento muy plausible, para determinar los pesos de red en cada una de las generaciones, se utiliza el algoritmo LMS puesto que este consigue determinar los pesos de manera rápida y optima. Una vez alcanzada la ultima generación, se aplica el algoritmo SVD para determinar de forma más exacta la configuración de pesos definitiva, el algoritmo SVD, siendo costoso del punto de vista computacional, se evita aplicarlo repetidamente y de esta manera, se aprovechan las cualidades de cada una de estas técnicas, minimizando sus inconvenientes.

3.5.3.2. Estudio del coeficiente de aprendizaje

En los métodos de descenso en gradiente convencionales, se suele mantener el valor del coeficiente de aprendizaje constante a lo largo del proceso de entrenamiento. Puesto que el rendimiento de un algoritmo cualquiera es muy sensible a la configuración de este coeficiente, como se ha mencionado previamente, si este tiene un valor demasiado alto, se provoca una oscilación y el algoritmo llega a ser inestable, y si este coeficiente es muy bajo, el algoritmo tardara mucho en converger. En cualquier caso, no es adecuado fijar el coeficiente de aprendizaje antes del proceso de entrenamiento, y por otro lado, notar que su valor óptimo varía a lo largo del proceso a medida que el algoritmo recorre el espacio de búsqueda.

Se puede optimizar el rendimiento del algoritmo de mínimos cuadrados si se configura un coeficiente de aprendizaje variable durante el proceso de aprendizaje. Así pues, un coeficiente adecuado trataría de aplicar un paso de aproximación lo más grande posible manteniendo estable el sistema. Un coeficiente de aprendizaje adaptativo es por lo tanto la solución idónea, y funcionaria del modo siguiente:

- Se calcula primero la salida de la red y el error.
- Se calculan en cada generación los pesos con el valor actual de η.
- Se calculan de nuevo la salida y el error.

En cada generación, si no se mejora el error, se descarta la nueva configuración y se disminuye el valor de η (multiplicándolo por $dec\_\eta = 0.7$), y si al contrario, se obtiene un resultado satisfactorio, se mantiene la nueva configuración y al coeficiente de aprendizaje se le aplica un incremento: $\eta = \eta \cdot inc\_\eta$, con $inc\_\eta = 1.05$.

Esta estrategia permite aumentar η, pero sin provocar inestabilidad del sistema, y se asegura obtener un valor optimo de η.

3.5.4 El error en la salida

Como ultima fase de la función de evaluación, se impone la determinación de un factor para medir la idoneidad de los individuos que se acaba de tratar. La medida del error es un buen indicador de la aptitud de una solución, además, y debido a su gran presencia en la bibliografía, permite realizar comparaciones entre métodos aplicados a un mismo problema y en condiciones similares.

En muchos casos se aplica el error cuadrático medio (*mean squared error*, MSE) que se escribe como:

$$\mathrm{MSE} = \frac{\sum_{i=1}^{n} \left(F(x_i) - f(x_i) \right)^2}{n} \tag{3.11}$$

Donde:

n es el número de datos de entrada.

$F(x_i)$ es la salida deseada.

$f(x_i)$ es la salida proporcionada por la red que está en evaluación para cada vector de entrada.

En nuestro trabajo, se medirá la raíz del error cuadrático medio normalizado (Normalized root mean squared error, NRMSE) que se expresa de la manera siguiente:

$$\text{NRMSE} = \sqrt{\frac{\text{MSE}}{\text{var}\left(F\left(x_i\right)\right)}} = \sqrt{\frac{\sum_{i=1}^{n}\left(F\left(x_i\right) - f\left(x_i\right)\right)^2}{\sum_{i=1}^{n}\left(F\left(x_i\right) - \overline{f}\right)^2}} \qquad (3.12)$$

Donde:

$\text{var}\left(F\left(x_i\right)\right)$ es la varianza de las salidas deseadas, y proporciona una estimación muy útil de un conjunto de vectores en una distribución normalizada.

$\overline{f}$ es la media de las salidas proporcionadas por la red.

3.6 La condición de parada

El proceso del algoritmo evolutivo precisa de una condición, que mientras no se cumpla, se repitan iteraciones en busca de una solución óptima. La manera más frecuente de parar este algoritmo es fijar un número máximo de generaciones [Van Dijk et al 2000]. En la bibliografía, existen heurísticas para detener un algoritmo genético, basadas en criterios como fijar un valor limite de optimización, y parar el proceso una vez alcanzado, existe también la posibilidad de exigir del algoritmo que pare si en un número determinado de generaciones no se mejora el resultado de manera sustantiva. Todas las condiciones descritas tienen un inconveniente, y es que precisan establecer un valor empírico, sea el número de generaciones máximo, sea una cuota de mejora en cada generación.

En este trabajo, aplicamos un criterio de parada que depende en igual modo del número de generaciones máximo y en la optimización continua del fitness del sistema.

Para aplicar estos criterios, se ha realizado un número de ejecuciones en las que se trata de aproximar diferentes funciones de una y dos dimensiones. El objetivo es determinar

un umbral máximo de generaciones que, a partir del cual el algoritmo no realiza ninguna optimización.

Una vez establecido este valor considerado como máximo, se inserta un segundo criterio basado en la tasa de optimización en las *i* anteriores generaciones, siendo *i* un número establecido con relatividad al número máximo de generaciones y se para el algoritmo si el valor de esta tasa es mínimo. La implantación de este criterio es bastante sencilla puesto que solamente implica una operación en cada iteración proporcionando una idea sobre la optimización realizada en las últimas generaciones.

Este razonamiento se ve apoyado por el hecho de que la implantación de 2 criterios basados en estudios empíricos es obviamente más fiable que la designación un único valor del que dependería en este caso la parada del sistema construido.

Tras aplicar el algoritmo propuesto a las funciones propuestas en el capitulo 4, los resultados muestran que no existe un umbral critico a partir del cual no se realiza una optimización del error, pero que en todos los casos, a la milésima generación se llega sin esperanzas de mejora, y por lo tanto este valor se puede establecer como valor tope del número de generaciones.

Del mismo modo que se ensaya el limite de generaciones tras el cual no se obtienen mas optimizaciones, se estudia en cada ejecución las porciones del proceso que no conocen una mejoría del resultado y en los cuales la curva del fitness o del error permanece plana, este estudio nos lleva a establecer el número de generaciones necesarias para parar el algoritmo sin llegar al limite máximo en 50 generaciones. Además, y gracias al planteamiento inicial en particular y al modo de funcionamiento del sistema propuesto que evita realizar convergencias prematuras, se evita caer en una parada prematura del algoritmo.

3.7 El algoritmo de minimización de error

Hasta el momento, se han aplicado metodologías para la aproximación de una función cualquiera mediante un algoritmo evolutivo que genera una red neuronal compuesta por RBFs. Esta estrategia consigue optimizar de una manera muy fiable la función objetivo,

eludiendo los principales problemas en los que se pueda caer, tal como una convergencia prematura hacia soluciones incompletas, óptimos locales, etc...

Aplicados en condiciones soportables de punto de vista computacional, tanto el algoritmo genético como la red de funciones RBF, y más concretamente, los valores de los parámetros que se usan como el tamaño de la población, el número de generaciones por un lado y el número de neuronas por otro, estas metodologías no aseguran alcanzar un óptimo global con una alta precisión. Y el razonamiento que hemos realizado en este capitulo tiene como principal objetivo, a parte de la introducción de nuevas funciones RBF entre otras aportaciones, trata de descubrir el perfecto equilibrio entre una estructura fiable y adecuada, y una carga en coste computacional y complejidad aceptables.

Para la fase que viene a continuación, se aplicaran métodos que anteriormente y en problemas convencionales, se podían aplicar como estrategias suficientes para una tarea de optimización, sin embargo en el caso nuestro con el objetivo de presentar un sistema robusto, y tratando de estudiar todas las situaciones que se puedan presentar, se les consideraran como complementarios a lo que se viene haciendo en este trabajo.

Estos métodos a los que se llama, algoritmos de minimización del error, tienen ventajas e inconvenientes de implementación. Por un lado, son capaces de realizar una búsqueda fina dentro de un intervalo determinado para alcanzar la mejor solución posible o al menos aproximarlo, y por otro, no son una garantía para implementarlas directamente desde el principio ya que son extremadamente costosas de punto de vista computacional, debido justamente a que su desarrollo preciso implica que en cada iteración sus alteraciones son pequeñas.

Mirando el trabajo realizado por el algoritmo evolutivo como una fase previa a la implementación de un algoritmo de minimización de error, se justifica el limite que se le impone en cuanto al número de generaciones y al número de tamaño de la población, y se puede considerar la solución propuesta por este algoritmo evolutivo, aun siendo válida, como una solución intermedia en el camino hacia la mejor solución posible.

En este trabajo se ha implementado un algoritmo de Levenberg-Marquardt, representa una extensión del algoritmo de Gauss-Newton, realiza una aproximación del error de una red neuronal, y se exprime de la manera siguiente:

$$\Delta w = -\left[\sum_{i=1}^{n} J_n{}^T J + \lambda I\right]^{-1} \nabla E \tag{3.13}$$

Donde J_n es la matriz jacobiana del vector de error evaluado en cada elemento de la red, λ es un parámetro variable, y $\nabla E(w)$ es el gradiente del error.

A lo largo del proceso, y se según si en cada iteración el error de la red es optimizado o no, el parámetro λ es dividido o multiplicado por un factor α como muestra el esquema del algoritmo de Levenberg-Marquardt:

1. Calcular el error para cada vector de entrada.
2. Calcular el valor de $err = \sum_{i=1}^{n} E(w)^T E^n(w)$ la matriz jacobiana para cada vector de entrada.
3. Calcular Δw.
4. Calcular $err\_act = \sum_{i=1}^{n} E(w+\Delta w)^T E^n(w+\Delta w)$.
5. Si:
   a. $err \le err\_act$, $\lambda = \lambda \cdot \alpha$.
   b. $err\_act < err$, $\lambda = \lambda / \alpha$ y $w + \Delta w$.

Algoritmo 3.2: *Algoritmo de Levenberg-Marquardt implantado en nuestro sistema.*

Los valores de λ y α se han fijado en 0.01 y 4 respectivamente.

Capítulo 4/
Resultados experimentales

En los capítulos anteriores, se han descrito, tanto el algoritmo S-RBF que se implanta en este trabajo, como las técnicas matemáticas y bio-inspiradas en las que se basa, se han discutido, las metodologías propuestas en trabajos anteriores, los razonamientos que llevan a la elección de cada heurística en lugar de otra, y los estudios empíricos que se han realizado en este trabajo para la obtención de diferentes valores de los parámetros que entran en la construcción de nuestro sistema.

En este capítulo, se evalúa el rendimiento del algoritmo SRBF**,** aplicándolo a diferentes tipos de funciones, tanto unidimensionales como bidimensionales de variables independientes. Como paso previo, se realizara un estudio completo del rendimiento de este algoritmo aplicado a varias funciones "sintéticas", estas funciones de una y dos dimensiones, han sido introducidas por primera vez en este trabajo, y sirven a comprobar la robustez de nuestro algoritmo frente a uno convencional.

4.1 Introducción

El proceso de simulaciones que seguiremos a lo largo de este capítulo depende de una multitud de factores, así pues en el establecimiento del número de vectores de entrada

que formaran los conjuntos de entrenamiento y los conjuntos de test y debido a que, para cada función utilizada, se presentara una tabla comparativa con otros trabajos en la materia, nos atemos a utilizar la misma configuración o en su defecto una similar para garantizar la coherencia de las comparaciones.

Para recoger la información necesaria sobre los indicadores del comportamiento de nuestro algoritmo, se necesita la ejecución del mismo un número determinado de veces, gracias al hecho de ser especialmente diseñado para la tarea de aproximación de funciones, este número de ejecuciones se ve reducido, obteniendo una idea fiable sobre la credibilidad del resultado sin tener que ejecutarlo muy repetidamente.

Una parte principal de este capítulo se dedica a realizar comparaciones con los resultados de otros trabajos en el campo de la aproximación de funciones, para ello, hemos de tener en cuenta varios parámetros, como pueden ser la complejidad de cada método aplicado o el coste computacional que suponen.

El conjunto de funciones a las que se aplica el algoritmo SRBF se consideran idóneas para los métodos de aproximación mediante RBFs, en la primera fase, el conjunto de funciones utilizado es variables independientes, a continuación se recurre a unas funciones unidimensionales y bidimensionales, ya utilizadas por autores anteriores.

Para evaluar los resultados obtenidos se ha tenido en cuenta dos factores: la eficiencia de los modelos generados y la complejidad de estos. Para medir la complejidad, dos parámetros han sido utilizados los parámetros siguientes:

- *m* que indica el número de neuronas que forman la red en el caso de la RBFNs y en el caso de los sistemas difusos, *m* se refiere al numero de reglas utilizadas por el sistema, implicando que una regla “si-entonces” dentro de un sistema difusa es el equivalente de una función de base radial dentro de una red neuronal artificial.
- n_p indica el número de parámetros libres que hay que establecer para definir un modelo dado. En el caso de una RBFN, n_p es el número de neuronas multiplicado por el número de coordenadas libres que fijan cada neurona, es decir: el peso, el radio, y el número de coordenadas de su centro, y permitirá

mediante la comparación a otros resultados de trabajos anteriores, la conveniencia de la implantación de tal parámetro.

Para comprobar la robustez del algoritmo, se incluye una comparación respecto a los resultados obtenidos en trabajos realizados recientemente en este mismo departamento. Asimismo, Pomares [Pomares, 2000] desarrolló un algoritmo de identificación de sistemas difusos, aplicable al problema de aproximación de funciones. Éste algoritmo trabaja con una tabla de reglas definidas que se obtiene mediante un algoritmo iterativo en el que se añade complejidad a la tabla de reglas hasta que se alcanza el criterio de parada.

González [González, 2001], también propone un método para el diseño de RBFNs. Se trata de un algoritmo genético donde se mantiene una población en la que cada individuo codifica una RBFN completa. Esto implica que, en cada generación, se pueden estar tratando varias decenas de redes a la vez, esta característica dice mucho de la potencia del sistema pero a la vez conlleva un gran coste computacional y una complejidad excesiva, sobre todo teniendo en cuenta las técnicas numéricas que se manejan como la aplicación de algoritmos para optimización de los pesos de la red y para minimización del error como el SVD, OLS, o el de Levenberg-Marquardt en cada generación.

En el trabajo de Rivas [Rivas, 2003], se presenta un típico algoritmo genético para el diseño de RBFNs. Se mantiene una población de individuos, donde cada uno de ellos representa una red. Estos individuos van evolucionando mediante la aplicación de un conjunto de operadores siguiendo una típica estrategia genética. Como resultados se ofrecen valores medios del error obtenido y el número de funciones base que se ha utilizado en las redes.

Rivera en su trabajo de tesis doctoral [Rivera, 2003] presenta una arquitectura hibrida de técnicas bio-inspiradas con una red neuronal en el nivel inferior de esta arquitectura que se encarga del procesamiento de los datos del problema mientras en el nivel siguiente se aplica una técnica basada en computación evolutiva cuya labor consiste en configurar los parámetros de la red.

Puesto que el algoritmo propuesto en este trabajo converge hacia soluciones similares al efectuar varias ejecuciones, gracias sobre todo a las técnicas utilizadas para guiar la evolución del sistema, se ha comprobado con satisfacción, por lo tanto, que las desviaciones estándar son bastante pequeñas.

Se han elaborado Las tablas de resultados a partir de 10 ejecuciones del algoritmo SRBF**.** En las tablas de resultados, se recogen los valores del NRMSE de la mejor ejecución, el de la peor ejecución, el valor medio de los resultados obtenidos, y la desviación estándar de estas 10 ejecuciones tanto para el conjunto de entrenamiento como par el de test. En todas las tablas se hacen simulaciones con el algoritmo S-RBF, y otro convencional.

Todas las ejecuciones se han realizado en un ordenador personal de procesador AMD 1.2GHz, utilizando el sistema operativo Windows® XP y bajo MATLAB® 6.0.

4.2 Resultados experimentales

A continuación detallamos los resultados de nuestro algoritmo. Los experimentos son distribuidos en 3 fases que son, funciones sintéticas, funciones de una dimensión, funciones de 2 dimensiones.

4.2.1 Funciones sintéticas

Estas funciones que se introducen por primera vez en este trabajo, son construidas a partir de la unión de un conjunto de sencillas funciones para emular las características de las funciones objetivo utilizadas en los problemas de aproximación de funciones y nos sirven para contrastar los resultados obtenidos por el algoritmo S-RBF frente a los de un algoritmo convencional. La implantación de este tipo de funcione coge una especial importancia cuando en el caso del algoritmo SRBF, puesto que este algoritmo tiene como principal característica la interpretación novedosa de cómo tratar los conjuntos de vectores de entrada a la altura de la capa oculta del sistema neuronal aplicando operadores T-conorma.

Las funciones sintéticas que se utilizan son de una y dos dimensiones, parten de la estructura básica de las RBFs, y se pueden expresar bajo la formula siguiente:

$$f\left(x_1, x_2, ..., x_k\right) = \sum_{k=1}^{dim} \sum_{i=1}^{n} w_i . exp\left(-\frac{\left\| x_k - c_i \right\|^2}{2\sigma_i^2} \right) \tag{4.1}$$

Los parámetros w_i, c_i, y σ_i son parámetros libres, n es el número de funciones base utilizadas.

Se caracterizan por tener unas variables independientes, y son adaptables al número de dimensiones en las que se quiera desarrollar el algoritmo en cuestión, en este trabajo se expondrán los resultados para $dim = 1$ y $dim = 2$ es decir: funciones sintéticas unidimensionales y bidimensionales, para poder resaltar los resultados experimentales acompañados por los aspectos gráficos de dichas funciones, sin embargo, y de la misma manera se pueden utilizar funciones de dimensiones superiores.

4.2.1.1 Funciones sintéticas de una dimensión

Para estas funciones, los conjuntos de entrenamiento están formados de 150 puntos repartidos de forma equidistante en el intervalo [-1,1], mientras que los conjuntos de test, están definidos por otros 150 puntos aleatorios, equidistribuidos, dentro del espacio de desarrollo de la función.

Las funciones iniciales que se tratara de aproximar se han escogido, a partir de la ecuación (4.1) y estableciendo los parámetros libres de centro, radio y peso de la forma siguiente:

$$sintetica\_1d\_4n\left(x\right) = \sum_{i=1}^{4} \varphi_i \tag{4.2}$$

$$sintetica\_1d\_6n\left(x\right) = \sum_{i=1}^{6} \varphi_i \tag{4.3}$$

$$sintetica\_1d\_8n\left(x\right) = \sum_{i=1}^{8} \varphi_i \tag{4.4}$$

$$sintetica\_1d\_12n(x) = \sum_{i=1}^{12} \varphi_i \tag{4.5}$$

Donde φ_i se escribe de la forma siguiente:

$$\varphi_i = w_i . exp\left(-\frac{\|x - c_i\|^2}{2\sigma_i^2} \right) \qquad x \in [-1,1] \tag{4.6}$$

Y los vectores de parámetros tienen los valores siguientes:

$$\begin{aligned} c_{12,1} &= [0.75; 0.20; -0.25; -0.70; 0.50; -1; 0; 1; -0.50; -0.10; 0.35; 0.85]^T \\ \sigma_{12,1} &= [0.20; 0.12; 0.25; 0.10; 0.18; 0.11; 0.50; 0.05; 0.08; 0.04; 0.12; 0.22]^T \\ w_{12,1} &= [1; -0.50; 0.33; 0.66; -0.50; 1; -0.80; -1; -0.70; 0.78; 0.85; -0.65]^T \end{aligned} \tag{4.7}$$

Los valores de los parámetros libres se han escogido de manera tal que las funciones en cuestión no tengan ninguna periodicidad ni simetría aparentes y que de alguna forma tengan una curva con el solapamiento conocido de las funciones objetivo habitualmente utilizadas para el entrenamiento de loa algoritmos evolutivos.

La figura 4.1, representa la primera de las funciones construidas a partir de la ecuación (4.1).

Se realizan las aproximaciones a la función descrita utilizando un número de RBFs creciente, tanto con un algoritmo convencional como con el algoritmo que proponemos en este trabajo.

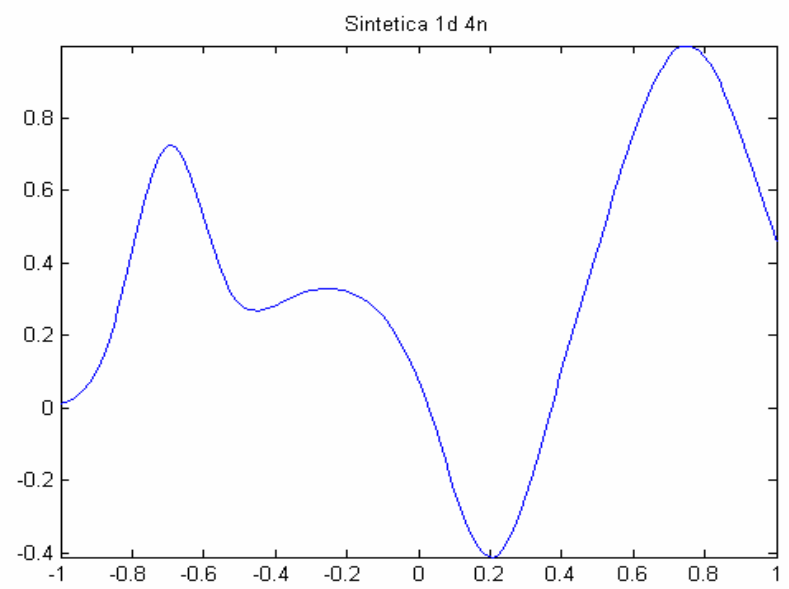


Figura 4.1: *Función sintética de una dimensión descrita en la ecuación (4.2).*

Algor.	*RBFs*	*Entrenamiento NRMSE*				*Test NRMSE*			
		Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
Red neuronal convencional	4	0.1738	0.2433	0.1997	0.0240	0.1774	0.2709	0.2012	0.0298
	6	0.0639	0.1173	0.0887	0.0237	0.0657	0.1341	0.0968	0.0217
	8	0.0451	0.0682	0.0553	0.0078	0.0467	0.1008	0.0693	0.0207
	10	0.0212	0.0388	0.0302	0.0077	0.0229	0.0387	0.0320	0.0062
	12	0.0109	0.0241	0.0153	0.0053	0.0127	0.0277	0.0174	0.0044
Algoritmo propuesto	4	0.0382	0.1692	0.0853	0.0209	0.0653	0.1658	0.1150	0.0372
	6	0.0239	0.0848	0.0583	0.0234	0.0174	0.0851	0.0615	0.0183
	8	0.0154	0.0432	0.0302	0.0103	0.0145	0.0436	0.0280	0.0092
	10	0.0123	0.0208	0.0168	0.0028	0.0116	0.0210	0.0163	0.0029
	12	0.0059	0.0115	0.0089	0.0022	0.0061	0.0109	0.0086	0.0016

Tabla 4.1: R*esultados del algoritmo* SRBF *aplicado a una función sintética descrita en la ecuación (4.2), y comparación con los resultados obtenidos mediante la aplicación de una red neuronal convencional*

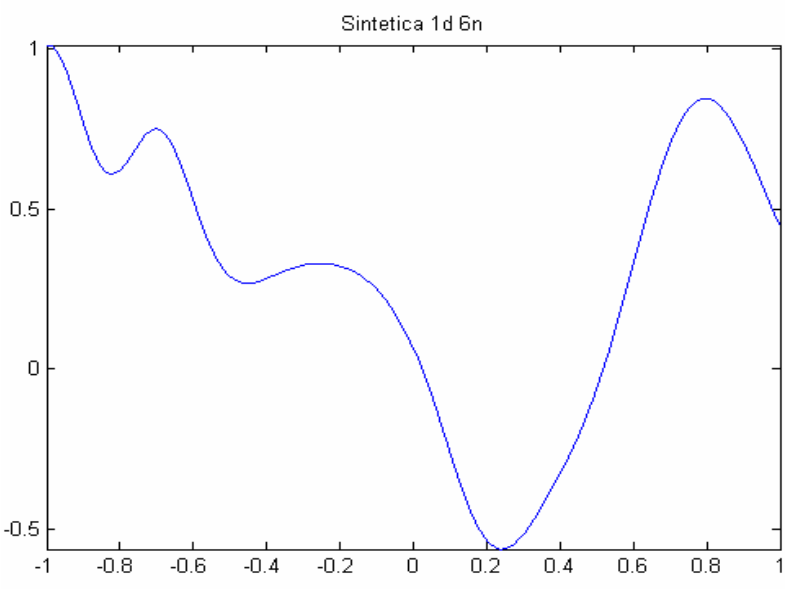


Figura 4.2: *Función sintética de una dimensión descrita en la ecuación (4.3).*

Algor.	*RBFs*	*Entrenamiento NRMSE*				*Test NRMSE*			
		Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
Red neuronal convencional	6	0.1095	0.1346	0.1264	0.0075	0.1069	0.1454	0.1223	0.0158
	8	0.0573	0.1061	0.0730	0.0145	0.0594	0.0873	0.0729	0.0106
	10	0.0309	0.0443	0.0372	0.0061	0.0339	0.0456	0.0393	0.0051
	12	0.0130	0.0323	0.0186	0.0058	0.0129	0.0309	0.0224	0.0073
	16	0.0027	0.0041	0.0033	4.2E-4	0.0027	0.0044	0.0034	6.9E-4
Algoritmo propuesto	6	0.0685	0.1012	0.0837	0.0124	0.0656	0.1054	0.0861	0.0118
	8	0.0229	0.0522	0.0729	0.0106	0.0216	0.0581	0.0427	0.0104
	10	0.0073	0.0237	0.0173	0.0054	0.0098	0.0248	0.0169	0.0051
	12	0.0034	0.0102	0.0072	0.0023	0.0047	0.0110	0.0082	0.0021
	16	0.0007	0.0018	0.0015	3.5E-4	0.0009	0.0018	0.0013	2.7E-4

Tabla 4.2: R*esultados del algoritmo* SRBF *aplicado a una función sintética descrita en la ecuación (4.3), y comparación con los resultados obtenidos mediante la aplicación de una red neuronal convencional.*

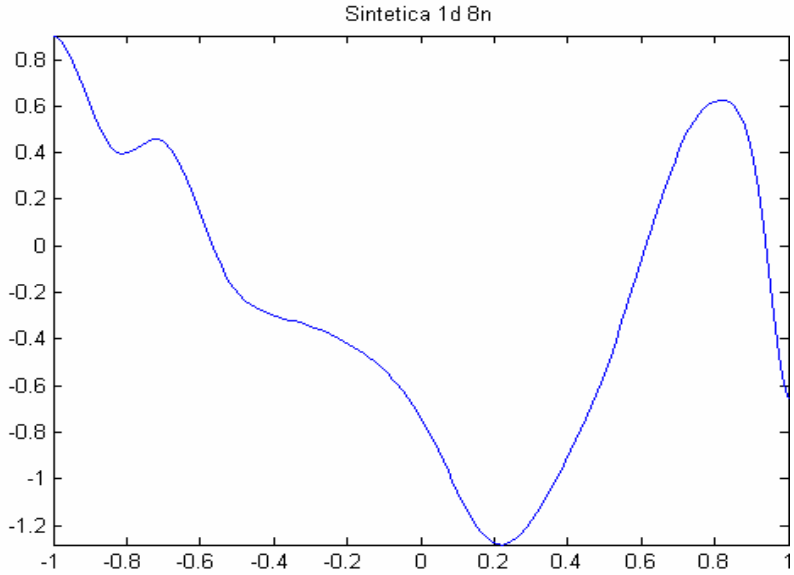


Figura 4.3. *Función sintética de una dimensión descrita en la ecuación (4.4).*

Algor.	*RBFs*	*Entrenamiento NRMSE*				*Test NRMSE*			
		Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
Red neuronal convencional	8	0.0672	0.0765	0.0706	0.0028	0.0670	0.0818	0.0711	0.0049
	10	0.0470	0.0634	0.0554	0.0070	0.0499	0.0602	0.0539	0.0043
	12	0.0307	0.0377	0.0326	0.0017	0.0300	0.0386	0.0327	0.0024
	16	0.0078	0.0129	0.0099	0.0016	0.0079	0.0134	0.0104	0.0013
	20	0.0027	0.0042	0.0035	4.9E-4	0.0027	0.0042	0.0038	3.7E-4
Algoritmo propuesto	8	0.0322	0.0508	0.0454	0.0039	0.0353	0.0523	0.0482	0.0073
	10	0.0236	0.0416	0.0338	0.0082	0.0240	0.0422	0.0365	0.0053
	12	0.0134	0.0272	0.0200	0.0043	0.0141	0.0285	0.0213	0.0041
	16	0.0019	0.0074	0.0050	0.0017	0.0021	0.0079	0.0052	0.0016
	20	0.0002	0.0026	0.0014	7.7E-4	0.0004	0.0029	0.0014	5.6E-4

Tabla 4.3: R*esultados del algoritmo* SRBF *aplicado a una función sintética descrita en la ecuación (4.4), y comparación con los resultados obtenidos mediante la aplicación de una red neuronal convencional.*

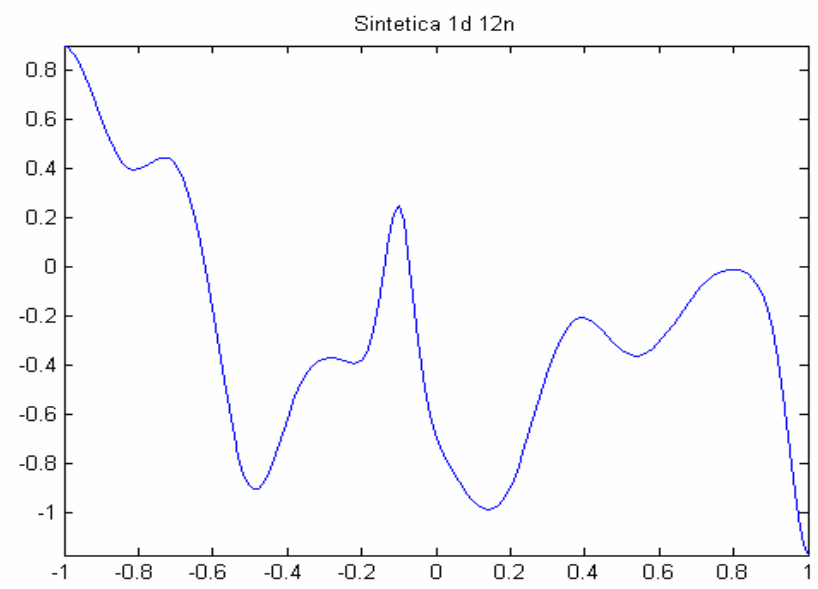


Figura 4.4: *Función sintética de una dimensión descrita en la ecuación (4.5).*

Algor.	*RBFs*	*Entrenamiento NRMSE*				*Test NRMSE*			
		Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
Red neuronal convencional	12	0.1063	0.1894	0.1288	0.0256	0.1137	0.1981	0.1456	0.0238
	14	0.0852	0.1120	0.0970	0.0087	0.0843	0.1175	0.0985	0.0086
	16	0.0775	0.984	0.0851	0.0071	0.0774	0.0969	0.0844	0.0060
	20	0.0304	0.0643	0.0459	0.0126	0.0294	0.0644	0.0435	0.0114
	24	0.0143	0.0408	0.0263	0.0113	0.0171	0.0395	0.0254	0.0074
Algoritmo propuesto	12	0.0663	0.1070	0.0897	0.0157	0.0843	0.1097	0.1021	0.0077
	14	0.0519	0.0775	0.0686	0.0097	0.0565	0.0761	0.0672	0.0083
	16	0.0457	0.0692	0.0566	0.0080	0.0453	0.0684	0.0583	0.0083
	20	0.0202	0.0331	0.0259	0.0044	0.0215	0.0346	0.0287	0.0050
	24	0.0100	0.0150	0.0126	0.0021	0.0105	0.0154	0.0127	0.0018

Tabla 4.4: R*esultados del algoritmo* SRBF *aplicado a una función sintética descrita en la ecuación (4.5), y comparación con los resultados obtenidos mediante la aplicación de una red neuronal convencional.*

Comparando los resultados de de los conjuntos de entrenamiento y de test, la primero conclusión que se puede sacar es que son bastante similares en su conjunto lo que demuestra que el algoritmo propuesto generaliza de manera muy optima.
Por otra parte, las diferencias entre los resultados obtenidos aplicando un algoritmo convencional y los obtenidos mediante nuestro algoritmo son bastante elocuentes, y resaltan de manera significativa la robustez del algoritmo propuesto en esta memoria.
Los valores de la desviación estándar, están dentro de los intervalos lógicos de los trabajos dirigidos a este tipo de tareas de aproximación de funciones.

4.2.1.2 Funciones sintéticas de dos dimensiones

Para estas funciones los conjuntos de entrenamiento están formados por 400 puntos distribuidos en una malla de 20 por 20 elementos equidistantes en el intervalo [-1,1].
Se establecen los parámetros de las funciones objetivo a partir de la ecuación (4.1):

$$sintetica\_2d\_4n(x_1,x_2) = \sum_{k=1}^{2}\sum_{i=1}^{4}\varphi_{ik} \tag{4.8}$$

$$sintetica\_2d\_6n(x_1,x_2) = \sum_{k=1}^{2}\sum_{i=1}^{6}\varphi_{ik} \tag{4.9}$$

$$sintetica\_2d\_8n(x_1,x_2) = \sum_{k=1}^{2}\sum_{i=1}^{8}\varphi_{ik} \tag{4.10}$$

$$sintetica\_2d\_12n(x_1,x_2) = \sum_{k=1}^{2}\sum_{i=1}^{12}\varphi_{ik} \tag{4.11}$$

Donde φ_{ik} se escribe de la forma siguiente:

$$\varphi_{ik} = \sum_{k=1}^{2} w_i . exp\left(-\frac{\left\| x_k - c_{ki} \right\|^2}{2\sigma_i^2} \right) \qquad x_k \in [-1,1] \tag{4.12}$$

Los parámetros se extraen de las formulas siguientes:

$$c_{12,2} = \begin{bmatrix} 0.75; 0.20; -0.25; -0.70; 0.50; -1; 0; 1; 0.75; 0.25; -0.85; -0.75 \\ 0.75; 0.20; -0.25; -0.70; 0.50; -1; 0; 1; -0.25; -0.15; 0.75; 0.25 \end{bmatrix}^T$$
$$\sigma_{12,1} = \left[0.20; 0.12; 0.25; 0.10; 0.18; 0.11; 0.50; 0.05; 0.80; 0.17; 0.50; 0.05\right]^T \quad (4.13)$$
$$w_{12,1} = \left[1; -0.50; 0.33; 0.66; -0.50; 1; -0.80; -1; -1; 0.70; -0.40; 0.30\right]^T$$

Se vuelve a comprobar la alta potencia del algoritmo propuesto en un ejemplo bastante más complicado que el primero.

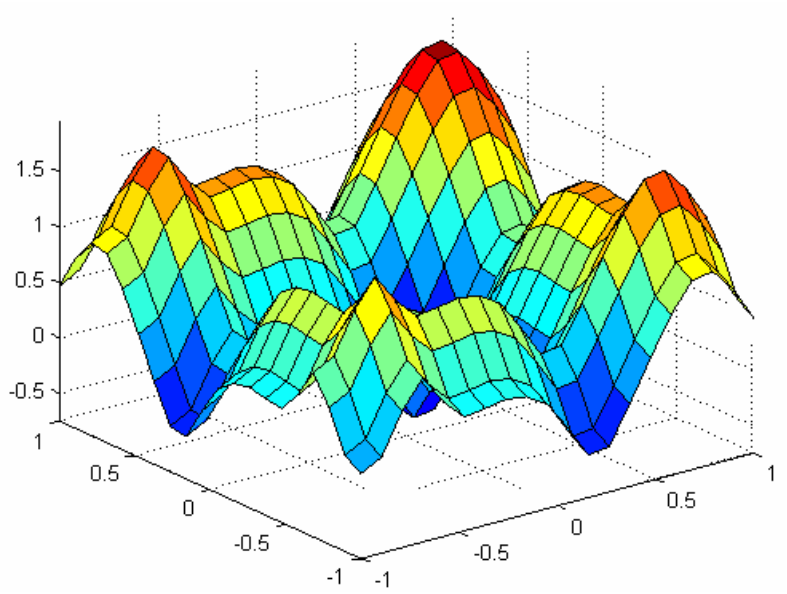


Figura 4.5. *Función sintética de dos dimensiones descrita en la ecuación (4.8).*

Algor.	*RBFs*	*Entrenamiento NRMSE*				*Test NRMSE*			
		Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
Red neuronal convencional	4	0.5369	0.6169	0.5715	0.0208	0.5344	0.6217	0.5721	0.0313
	6	0.4337	0.5091	0.4797	0.0242	0.4228	0.5125	0.4713	0.0262
	8	0.3574	0.4334	0.4009	0.0256	0.3688	0.4384	0.4065	0.0257
	10	0.3229	0.4256	0.3752	0.0257	0.3453	0.4355	0.3722	0.0225
	12	0.3116	0.3746	0.3401	0.0228	0.3252	0.3703	0.3441	0.0206
Algoritmo propuesto	4	0.1930	0.2963	0.2495	0.0320	0.1813	0.3027	0.2629	0.0288
	6	0.1185	0.2055	0.1685	0.0288	0.1228	0.2123	0.1691	0.0274
	8	0.0886	0.1633	0.1290	0.0233	0.0910	0.1614	0.1311	0.0218
	10	0.0679	0.1504	0.1032	0.0251	0.0665	0.1462	0.1004	0.0229
	12	0.0447	0.1159	0.0787	0.0176	0.0524	0.1233	0.0812	0.0213

Tabla 4.5: R*esultados del algoritmo* SRBF *aplicado a una función sintética descrita en la ecuación (4.8), y comparación con los resultados obtenidos mediante la aplicación de una red neuronal convencional.*

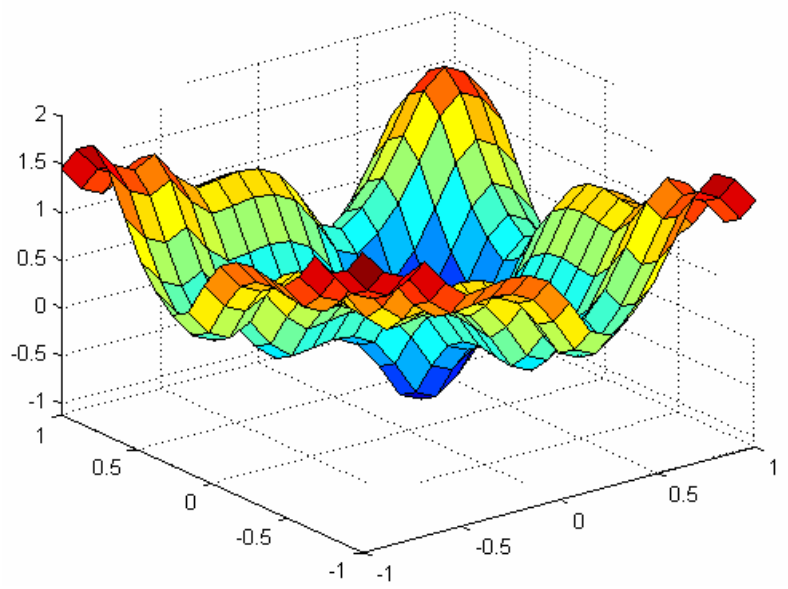


Figura 4.6. *Función sintética de dos dimensiones descrita en la ecuación (4.9).*

Algor.	*RBFs*	*Entrenamiento NRMSE*				*Test NRMSE*			
		Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
Red neuronal convencional	6	0.5838	0.6472	0.6192	0.0251	0.5816	0.6586	0.6197	0.0268
	8	0.4909	0.5425	0.5188	0.0175	0.4998	0.5568	0.5205	0.0223
	10	0.4204	0.4830	0.4585	0.0244	0.4298	0.4869	0.4533	0.0219
	12	0.3686	0.3993	0.3840	0.0173	0.3660	0.3980	0.3821	0.0126
	14	0.2522	0.2970	0.2718	0.0092	0.2507	0.3076	0.2747	0.0196
Algoritmo propuesto	6	0.1802	0.2750	0.2242	0.0365	0.1733	0.2742	0.2231	0.0337
	8	0.1445	0.2281	0.1935	0.0296	0.1445	0.2315	0.1864	0.0293
	10	0.0607	0.1189	0.0984	0.0170	0.0615	0.1247	0.0992	0.0193
	12	0.0405	0.0621	0.0492	00083	0.0426	0.0606	0.0497	0.0074
	14	0.0221	0.0370	0.0273	0.0062	0.0232	0.0376	0.0284	0.0052

Tabla 4.6: R*esultados del algoritmo* SRBF *aplicado a una función sintética descrita en la ecuación (4.9), y comparación con los resultados obtenidos mediante la aplicación de una red neuronal convencional.*

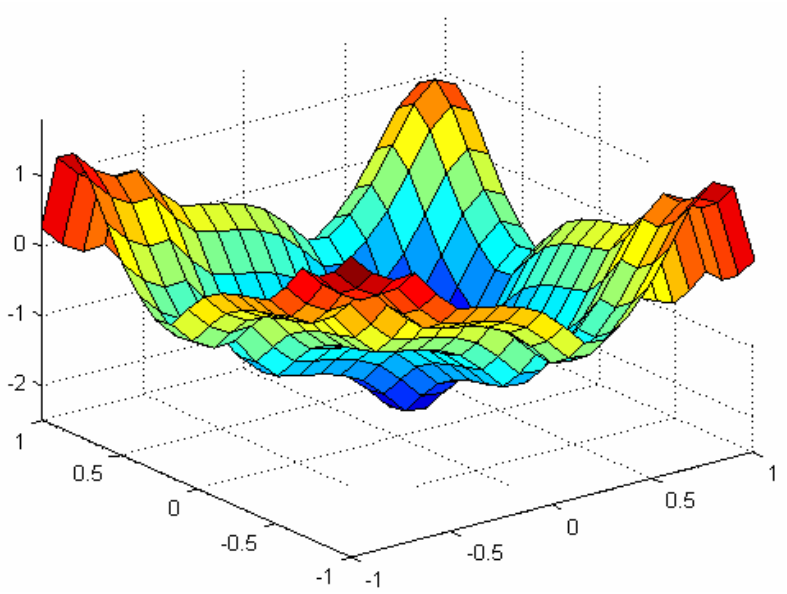


Figura 4.7: *Función sintética de dos dimensiones descrita en la ecuación (4.10).*

Algor.	*RBFs*	*Entrenamiento NRMSE*				*Test NRMSE*			
		Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
Red neuronal convencional	8	0.5869	0.6511	0.6221	0.0228	0.5947	0.6423	0.6184	0.0261
	10	0.5049	0.5730	0.5493	0.0261	0.5277	0.5729	0.5531	0.0214
	12	0.4118	0.4763	0.4388	0.0255	0.4293	0.4672	0.4422	0.0168
	14	0.3169	0.3673	0.3428	0.0236	0.3051	0.3627	0.3386	0.0181
	16	0.1973	0.2527	0.2364	0.0210	0.2130	0.2677	0.2389	0.0204
Algoritmo propuesto	8	0.2219	0.2639	0.2457	0.0172	0.2205	0.2719	0.2497	0.0160
	10	0.1563	0.2055	0.1762	0.0230	0.1637	0.2102	0.1819	0.0192
	12	0.0801	0.1241	0.1028	0.0166	0.1028	0.1420	0.1136	0.0163
	14	0.0338	0.0722	0.0571	0.0179	0.0403	0.0852	0.0614	0.0141
	16	0.0226	0.0474	0.0340	0.0113	0.0193	0.0661	0.0410	0.0082

Tabla 4.7: R*esultados del algoritmo* SRBF *aplicado a una función sintética descrita en la ecuación (4.10), y comparación con los resultados obtenidos mediante la aplicación de una red neuronal convencional.*

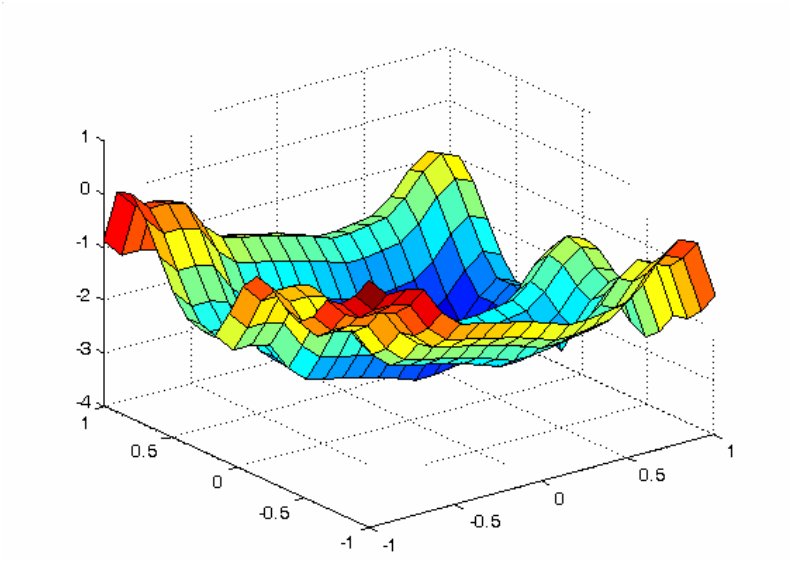


Figura 4.8: *Función sintética de dos dimensiones descrita en la ecuación (4.11).*

Algor.	*RBFs*	*Entrenamiento NRMSE*				*Test NRMSE*			
		Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
Red neuronal convencional	12	0.4603	0.5038	0.4723	0.0204	0.4548	0.4904	0.4660	0.0180
	14	0.3827	0.4146	0.3991	0.0130	0.3732	0.3961	0.3804	0.0172
	16	0.2857	0.3288	0.3144	0.0172	0.3037	0.3452	0.3247	0.0242
	18	0.2173	0.2723	0.2460	0.0223	0.2283	0.2603	0.2414	0.0170
	20	0.1394	0.1688	0.1536	0.0182	0.1538	0.1924	0.1629	0.0228
Algoritmo propuesto	12	0.1893	0.2279	0.2013	0.0145	0.2073	0.2583	0.2241	0.0151
	14	0.1131	0.1452	0.1292	0.0231	0.1187	0.1603	0.1356	0.0196
	16	0.0647	0.0923	0.0767	0.0152	0.0542	0.0829	0.0681	0.0122
	18	0.0159	0.0436	0.0301	0.0139	0.0227	0.0512	0.0343	0.0092
	20	0.0107	0.0230	0.0121	0.0049	0.0111	0.0283	0.0149	0.0087

Tabla 4.8: R*esultados del algoritmo* SRBF *aplicado a una función sintética descrita en la ecuación (4.11), y comparación con los resultados obtenidos mediante la aplicación de una red neuronal convencional.*

Los resultados obtenidos tras estas ejecuciones confirman que el algoritmo SRBF propuesto mejora en todas las situaciones el rendimiento obtenido por una algoritmo de aproximación convencional basado en redes RBF.

4.2.1.3 Conclusiones

La introducción de este tipo de funciones sintéticas para entrenar un algoritmo de aproximación de funciones es una de las novedades aportadas por este algoritmo. Los apartados anteriores en los que se han creado diferentes funciones de una y dos dimensiones, de complejidad variada han servido para obtener una idea muy relevante sobre el comportamiento del algoritmo SRBF aportado en este trabajo frente a un algoritmo convencional basado en un algoritmo evolutivo.

Los resultados reflejan de manera unánime una robustez y una fiabilidad altísimas tal y como muestran los valores de las desviaciones estándar los resultados, los resultados

obtenidos mejoran sustancialmente los resultados generados por un algoritmo convencional, mientras la poca diferencia entre valores de entrenamiento y valores de test demuestran la buena generalización del algoritmo SRBF.

Estos resultados permiten encarar la siguiente fase en la que aplicamos nuestro algoritmo a la aproximación de funciones previamente utilizadas en la biografía lo que nos permitirá realizar unas comparativas con los resultados obtenidos en trabajos que tratan el mismo problema que nosotros.

4.2.2 Funciones de una dimensión

Tras aplicar nuestro algoritmo a funciones sintéticas construidas por nosotros para una experimentación intermedia, en esta sección recurrimos a varias funciones previamente utilizadas en la biografía para comprobar algoritmo de aproximación de funciones. Para cada función escogida, se presentara una tabla con los resultados obtenidos en cada ejecución, así como una comparativa con otros trabajos realizados en condiciones parecidas o similares.

4.2.2.1 Funciones *wm1* y *wm2*

El algoritmo propuesto por Wang y Mendel [Wang & Mendel, 1992] es uno de los primeros trabajos que abordan la aproximación de funciones mediante un algoritmo aprendiente a partir de un conjunto de vectores de entrenamiento, la idea siendo generar un conjunto de reglas difusas, una por cada vector de entrada, e ir eliminando aquellas reglas que menos interfieren en el desarrollo del sistema. Sudkamp y Hamell [Sudkamp & Hamell, 1994] posteriormente, introdujeron una mejoría basada en la reducción del efecto del ruido en las consecuencias de las reglas difusas, asimismo plasmaron la teoría de que unas entradas similares deben producir unas salidas similares. Para ello, presentaron dos métodos como son el de “crecimiento por regiones” (Region Growing, RG) y el método de la “media ponderada” (Weighted Average, WA). Las funciones elegidas para comprobar estos algoritmos son:

$$wm_1(x) = x^3 \qquad x \in [-1,1] \qquad (4.14)$$

$$wm_2(x) = \sin(2\pi x) \qquad x \in [-1,1] \qquad (4.15)$$

Aplicamos el algoritmo SRBF con valores ascendentes del número de RBFs. Las figuras 4.9 y 4.10 representan los gráficos de estas funciones. Mientras las tablas 4.9 y 4.11 representan los resultados obtenidos, aplicando el algoritmo SRBF a un conjunto de entrenamiento formado por 150 puntos equidistribuidos en el espacio de entrada, y otro de test compuesto por 500 puntos distribuidos uniformemente en el mismo espacio.
Como mencionamos al principio, realizamos una comparación de los resultados generados por nuestro algoritmo respecto a otros trabajos anteriores.
De la tabla 4.10 relevamos que nuestro algoritmo mejora sustancialmente los resultados obtenidos mediante la utilización de sistemas difusos para la aproximación de la función *wm1*, incluso utilizando un número parámetros menor, este comportamiento tiene su explicación en la utilización de funciones gaussianas en nuestro algoritmo frente a las funciones triangulares aplicadas en los algoritmo basados en sistemas difusos, lo que provoca una interpolación suave y permite aproximar de mejor manera las funciones utilizadas. Respecto a los trabajos de basados en redes RBFNs, relevamos que nuestro algoritmo se sitúa en un nivel intermedio, mejorando los resultados de González [González, 2001] cuando se utilizan pocas RBFs, mientras que comparado a los resultados de Rivera [Rivera, 2003], nuestro algoritmo los mejora sustancialmente a partir de un número medio y medio alto de neuronas en nuestra red, este dato tiene su explicación en el hecho que el algoritmo de Rivera, interrumpe su desarrollo en un valor reducido de RBFs mientras que nuestro algoritmo continua optimizando la función objetivo.
En lo que se refiere a la función *wm2*, el algoritmo que proponemos se sitúa en los mismos niveles que los trabajos de González y Rivera, mejorando estos últimos cuando se utilizan pocas RBFs, además, consideramos recordamos que nuestro algoritmo es mucho menos costoso de punto de vista computacional.

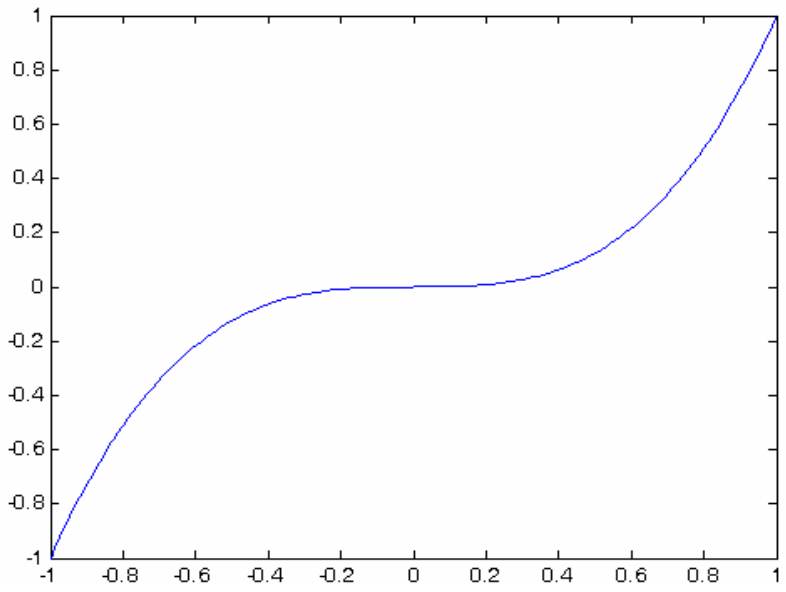


Figura 4.9: *Función wm1 descrita en la ecuación (4.14).*

RBFs	*Entrenamiento NRMSE*				*Test NRMSE*			
	Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
2	0.0144	0.0172	0.0155	9.4E-4	0.0142	0.0167	0.0159	0.0017
3	0.0101	0.0154	0.0129	0.0026	0.0097	0.0148	0.0129	0.0025
4	0.0014	0.0052	0.0030	0.0012	0.0016	0.0055	0.0032	0.0015
5	0.0003	0.0014	9.5E-4	3.1E-4	2.8E-4	0.0016	0.0010	4.3E-4
6	5.5E-5	4.3E-4	2.4E-4	1.2E-4	5.5E-5	4.4E-4	2.4E-4	1.6E-4
7	4.5E-5	3.4E-4	1.7E-4	8.6E-5	4.2E-5	3.3E-4	1.5E-4	7.8E-5
8	3.3E-5	1.0E-4	4.3E-5	2.9E-5	3.6E-5	1.0E-4	4.4E-5	2.7E-5
9	1.1E-6	1.0E-5	5.6E-6	3.3E-6	1.2E-6	9.7E-6	5.6E-6	3.2E-6
10	3.1E-7	3.9E-6	1.5E-6	1.0E-6	3.2E-7	3.9E-6	1.5E-6	9.5E-7

Tabla 4.9: R*esultados del algoritmo* SRBF *aplicado a la función wm1 descrita en la ecuación (4.14).*

Algoritmo		m	n_p	Error medio	NRMSE
[Wang & Mendel, 1992]		15	-	0.029	-
		25	-	0.018	-
Mejora de Wang y Mendel en [Sudkamp &Hamell, 1994]		15	-	0.079	-
		25	-	0.088	-
[Sudkamp & Hamell, 1994]	RG	15	-	0.044	-
		25	-	0.021	-
	WA	15	-	0.044	-
		25	-	0.021	-
[Pomares, 2000]		4	6	0.026	0.080
		6	10	0.011	0.032
		8	14	0.006	0.017
[González, 2001]		3	9	0.0042 ± 0.0005	0.0134 ± 0.001
		4	12	$0.0001 \pm 5.4E\text{-}5$	0.0004 ± 0.0002
		5	15	$2.1E\text{-}5 \pm 1.8E\text{-}5$	$6.8E\text{-}5 \pm 5.7E\text{-}5$
[Rivera, 2003]		3	9	0.0032 ± 0.0009	0.0115 ± 0.0029
		4	12	0.0012 ± 0.0004	0.0040 ± 0.0016
		5	15	0.0018 ± 0.0009	0.0064 ± 0.0037
Algoritmo propuesto		3	9	0.0038 ± 0.0010	0.0129 ± 0.0026
		4	12	$0.0009 \pm 5.3E\text{-}4$	0.0030 ± 0.0012
		5	15	$3.1E\text{-}4 \pm 1.7E\text{-}4$	$9.5E\text{-}4 \pm 3.1E\text{-}4$

Tabla 4.10: *Comparación de los resultados obtenidos para la Función wm1 respecto a otros algoritmo.*

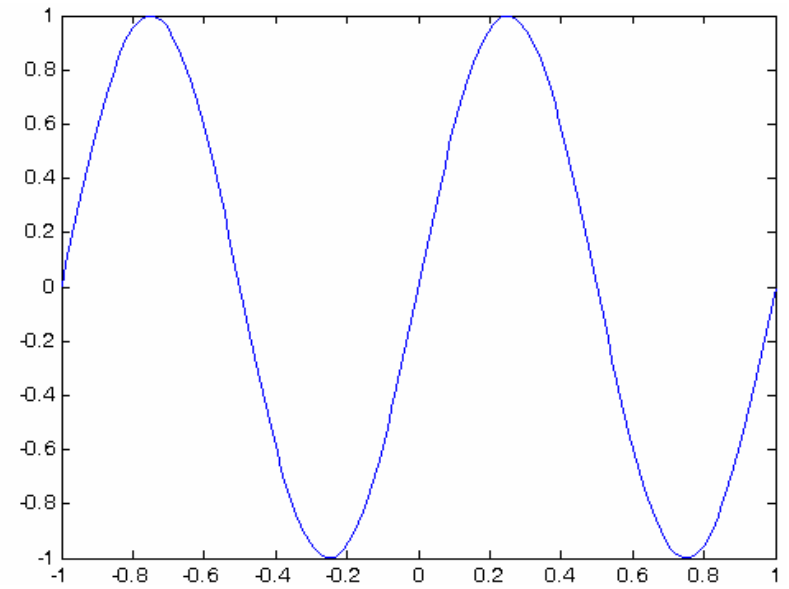


Figura 4.10: *Función wm2 descrita en la ecuación (4.15).*

RBFs	*Entrenamiento NRMSE*				*Test NRMSE*			
	Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
3	0.1780	0.1799	0.1786	0.0008	0.1780	0.1795	0.1785	0.0008
4	0.0253	0.0377	0.0341	0.0184	0.0258	0.0354	0.0338	0.0179
5	0.0229	0.0304	0.0286	0.0068	0.0221	0.0312	0.0290	0.0073
6	0.0023	0.0164	0.0087	0.0059	0.0029	0.0171	0.0092	0.0056
7	0.0031	0.0094	0.0060	0.0024	0.0031	0.0100	0.0064	0.0025
8	0.0018	0.0065	0.0041	0.0011	0.0016	0.0060	0.0038	0.0010
9	0.0016	0.0049	0.0032	0.0007	0.0016	0.0048	0.0032	0.0007
10	0.0009	0.0044	0.0027	0.0005	0.0010	0.0042	0.0026	0.0004

Tabla 4.11: R*esultados del algoritmo* SRBF *aplicado a la función wm2 descrita en la ecuación (4.15).*

Algoritmo		m	n_p	Error medio	NRMSE
[Wang & Mendel, 1992]		15	-	0.060	-
		25	-	0.026	-
Mejora de Wang y Mendel en [Sudkamp &Hamell, 1994]		15	-	0.071	-
		25	-	0.031	-
[Sudkamp & Hamell, 1994]	RG	15	-	0.131	-
		25	-	0.052	-
	WA	15	-	0.180	-
		25	-	0.082	-
[Pomares, 2000]		6	10	0.068	0.112
		8	14	0.0473	0.0823
		10	18	0.0262	0.0237
[González, 2001]		3	9	0.0582 ± 0.0003	$0.1169 \pm 4.0\text{E-}5$
		4	12	0.0107 ± 0.0111	0.0200 ± 0.0237
		5	15	0.0068 ± 0.0068	0.0134 ± 0.0173
		6	18	0.0028 ± 0.0013	0.0049 ± 0.0023
[Rivas,2003]		11	33	$1.2\text{E-}4 \pm 0.9\text{E-}4$	-
[Rivera, 2003]		3	9	0.1898 ± 0.0013	0.5071 ± 0.0003
		4	12	0.0227 ± 0.0033	0.0481 ± 0.0063
		5	15	0.0127 ± 0.0011	0.0310 ± 0.0015
		6	18	0.0006 ± 0.0004	0.0011 ± 0.0008
Algoritmo propuesto		3	9	0.0811 ± 0.0034	0.1786 ± 0.0008
		4	12	0.0158 ± 0.0096	0.0341 ± 0.0184
		5	15	0.0124 ± 0.0042	0.0286 ± 0.0068
		6	18	0.0036 ± 0.0027	0.0087 ± 0.0059

Tabla 4.12: *Comparativa entre diferentes algoritmos aplicados a la función wm2.*

4.2.2.2 Función *dick*

Esta función (4.16) representada en la figura 4.11 ha sido utilizada en [Dickerson & Kosko, 1996] para aplicar un sistema neuro-difuso con reglas elipsoidales y una combinación de distintos valores para los pesos de las reglas, así como diferentes métodos de aprendizaje.

$$dick(x) = 3x(x-1)(x-1.9)(x+0.7)(x+1.8) \qquad x \in [-2.1, 2.1] \qquad (4.16)$$

Mientras que Pomares utilizó para la aproximación de esta función un sistema difuso con funciones de pertenencia triangulares basado en un conjunto de reglas que se obtienen mediante un algoritmo iterativo en el que se añade complejidad a la tabla de reglas hasta alanzar un criterio de parada. Gracias al principio de equivalencia, podemos equiparar una función RBF con su peso asociado a una regla difusa.

La tabla 4.13 muestra los resultados obtenidos a partir de configuraciones cada vez más complejas del algoritmo SRBF, cuanto más aumenta el número de RBFs, disminuyen los errores medios y normalizados, si bien no tiene sentido utilizar más de 6 neuronas para aproximar esta función, puesto que a partir de este nivel, el aumento de coste computacional no es justificable.

Comparado con otros trabajos, destacamos que nuestro algoritmo mejora los resultados obtenidos por Dickerson con sus diferentes métodos mientras el algoritmo de Rivas [Rivas, 2003] consigue buenos resultados pero a costa de un coste computacional excesivo, debido al alto numero de parámetros utilizados.

Nuestros resultados asimismo mejoran considerablemente los obtenidos por algoritmos basados en RBFs en situaciones de pocas funciones base, mientras en los casos en los que se recurre a un mayor número de RBFs, el rendimiento del algoritmo SRBF se asemeja al de los demás algoritmos, lo que confirma el buen comportamiento de nuestro algoritmo respecto a otros trabajos.

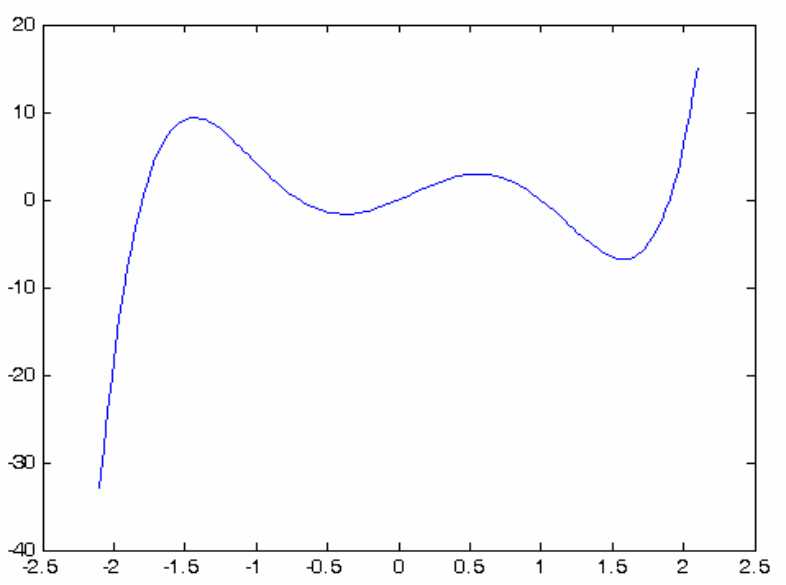


Figura 4.11: *Función dick descrita en la ecuación (4.16).*

RBFs	*Entrenamiento NRMSE*				*Test NRMSE*			
	Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
3	0.1585	0.1599	0.1592	8.4E-4	0.1575	0.1586	0.3578	9.2E-4
4	0.0952	0.1414	0.1123	0.0146	0.0948	0.1497	0.1175	0.0175
5	0.0815	0.1087	0.0916	0.0022	0.0783	0.1054	0.0933	0.0079
6	0.0011	0.0019	0.0014	7.4E-4	0.0010	0.0020	0.0014	9.5E-4
7	0.0012	0.0021	0.0015	7.2E-4	0.0012	0.0021	0.0015	7.2E-4
8	0.0008	0.0023	0.0013	9.2E-4	0.0008	0.0022	0.0013	8.1E-4
9	0.0009	0.0020	0.0014	5.2E-4	0.0010	0.0019	0.0013	4.7E-4
10	0.0008	0.0020	0.0013	5.8E-4	0.0010	0.0021	0.0014	8.2E-4

Tabla 4.13: *Resultados del algoritmo* SRBF *aplicado a la función dick descrita en la ecuación (4.16).*

Algoritmo			m	n_p	Error medio	NRMSE
[Dickerson & Kosko, 1996]	Distintos pesos de la red	$w_k = 1$	6	-	94.65	-
		$w_k = 1/v_k$			28.25	-
		$w_k = 1/v_k^2$			10.53	-
	Aprend. no superv.				7.927	-
	Aprend. superv.				3.069	-
[Pomares, 2000]			5	8	5.01	0.33
			6	10	1.35	0.17
			7	12	0.46	0.10
[González, 2001]			3	9	5.57 ± 0	0.3455 ± 0
			4	12	0.99 ± 0.49	0.1415 ± 0.0390
			5	15	0.30 ± 0.02	0.0797 ± 0.0023
[Rivas,2002]			8.3	24.9	0.05 ± 0.04	-
[Rivera, 2003]			3	9	3.57 ± 0	0.2839 ± 0
			4	12	1.27 ± 0.38	0.1671 ± 0.0285
			5	15	0.26 ± 0.09	0.0739 ± 0.0239
			6	18	0.0057 ± 0.01	0.0036 ± 0.0052
Algoritmo propuesto			3	9	2.23 ± 0.002	$0.1592 \pm 8.4\text{E-}4$
			4	12	0.82 ± 0.04	0.1123 ± 0.0146
			5	15	0.33 ± 0.07	0.0916 ± 0.0022
			6	18	$3.4\text{E-}3 \pm 0.002$	$0.0014 \pm 7.4\text{E-}4$

Tabla 4.14: Representación c*omparativa de diferentes algoritmos para la aproximación de la función dick.*

4.2.2.3 Función *nie*

Nie y Lee [Nie & Lee, 1996] desarrollaron tres algoritmos: P1, P2 y P3 dirigidos a la aproximación de funciones compuestos por un sistema difuso de 30 reglas. La medida de la robustez del sistema ha sido evaluada a partir de los valores del error cuadrático medio, estos algoritmos han sido aplicados a la función siguiente:

$$nie(x) = 3e^{-x^2}.\sin(\pi x) \qquad x \in [-3,3] \qquad (4.17)$$

En el sistema de construcción de sistemas difusos [Pomares. 2000], se ha utilizado implantaciones de 6, 7 y 10 reglas definidas mediante una partición triangular obteniendo mejores resultados tanto en error como en complejidad del sistema.
La tabla 4.15 muestra que los valores del error van mejorando a medida que aumenta el número de RBFs utilizadas.
La tabla de comparaciones 4.16, muestra un comportamiento óptimo del algoritmo propuesto frente a los sistemas presentados por Nie, mientras frente a los algoritmos basados en RBFs, destacamos que nuestro algoritmo mejora los a los de Rivera cuando se utiliza un número reducido de RBFs, y de Rivas, aun recurriendo este a un número de parámetros muy elevado.

4.2.2.4 Función *pom*

La función *pom* introducida por primera vez por Pomares [Pomares, 2000] para resaltar el efecto de una buena inicialización en el comportamiento de algoritmo dado. Esta función se escribe:

$$pom(x) = 3e^{-3x}.\sin(10\pi x) \qquad x \in [0,1] \qquad (4.18)$$

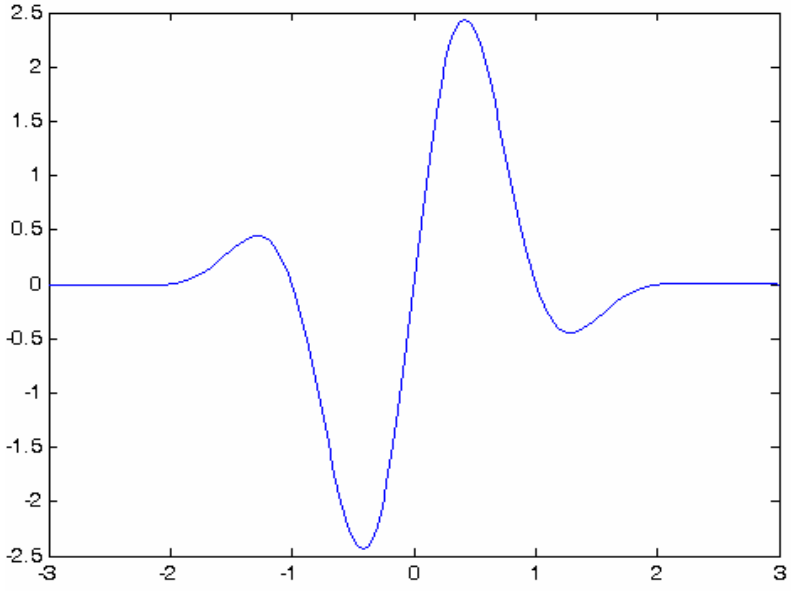


Figura 4.12: *Función nie descrita en la ecuación (4.17).*

RBFs	*Entrenamiento NRMSE*				*Test NRMSE*			
	Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
3	0.0554	0.0938	0.0736	0.0141	0.0554	0.1011	0.0759	0.0174
4	0.0159	0.0379	0.0269	0.0059	0.0144	0.0355	0.0254	0.0068
5	0.0086	0.0130	0.0109	0.0017	0.0090	0.0138	0.0114	0.0021
6	0.0049	0.0115	0.0085	0.0025	0.0047	0.0113	0.0080	0.0022
7	0.0032	0.0085	0.0048	0.0022	0.0033	0.0080	0.0045	0.0018
8	0.0025	0.0069	0.0036	9.2E-4	0.0024	0.0065	0.0035	7.1E-4
9	0.0017	0.0052	0.0031	4.1E-4	0.0019	0.0052	0.0032	4.3E-4
10	0.0008	0.0025	0.0015	5.6E-4	0.0010	0.0024	0.0017	4.9E-4

Tabla 4.15: R*esultados del algoritmo* SRBF *aplicado a la función nie descrita en la ecuación (4.17).*

Algoritmo		m	n_p	Error medio	NRMSE
[Nie, 1994]	P1/γ1	30	-	0.0114	-
	P1/γ2			0.0078	
	P2/γ1			0.0071	-
	P2/γ2			0.0068	-
	P3/γ1			0.0081	-
	P3/γ2			0.0082	-
[Pomares, 2000]		6	10	0.0113	0.111
		7	12	0.0092	0.099
		10	18	0.0024	0.051
[González, 2001]		3	9	0.0066 ± 0.0085	0.0647 ± 0.0601
		4	12	5.73E-5 $\pm$ 1.03E-5	0.0078 ± 0.0007
		5	15	2.87E-5 $\pm$ 3.1E-5	0.0048 ± 0.0032
		6	18	1.74E-5 $\pm$ 1.57E-5	0.0037 ± 0.0024
[Rivas,2002]		8.5	25.5	1.03E-5 $\pm$ 1.07E-5	-
[Rivera, 2003]		3	9	0.0145 ± 0.0046	0.1199 ± 0.0333
		4	12	6.5E-5 $\pm$ 1.0E-5	0.0083 ± 0.0006
		5	15	3.1E-5 $\pm$ 0.8E-5	0.0057 ± 0.0011
		6	18	0.1E-5 $\pm$ 0.1E-5	0.0010 ± 0.0007
Algoritmo propuesto		3	9	0.0086 ± 0.0021	0.0736 ± 0.0141
		4	12	4.2E-4 $\pm$ 8.5E-5	0.0269 ± 0.0059
		5	15	9.6E-5 $\pm$ 1.1E-4	0.0109 ± 0.0017
		6	18	6.3E-5 $\pm$ 1.4E-5	0.0085 ± 0.0025

Tabla 4.16: *Comparativa de diferentes algoritmos para la aproximación de la función nie.*

Esta función describe perfectamente la dependencia de la variabilidad respecto al valor de *x*, es decir, tomando el 0 como referencia, cuanto más se aleja de este punto menos variabilidad tiene la función *pom*, un buen algoritmo debería priorizar esta zona del espacio de búsqueda a la hora de colocar funciones aproximativas.

La figura 4.13 muestra tanto la función original, como su aproximación mediante un número creciente de RBFs, se comprueba de manera grafica la idea que explicamos anteriormente, la tabla de resultados 4.17 muestra numéricamente estos resultados.

En la tabla de comparaciones con otros algoritmos, se aprecia que el algoritmo SRBF que desarrollamos en este trabajo mejora los resultados obtenidos por Pomares, y más nítidamente los de Rivera, respecto a González, destacamos que, con unos resultados casi tan óptimos, nuestro algoritmo los mejora del punto de vista de la desviación estándar, lo que testifica de la buena generalización del algoritmo SRBF.

RBFs	*Entrenamiento NRMSE*				*Test NRMSE*			
	Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
4	0.2310	0.3159	0.2794	0.0415	0.2322	0.3205	0.2810	0.0394
5	0.1664	0.2309	0.1986	0.0456	0.1664	0.2300	0.1984	0.0456
6	0.1139	0.1700	0.1380	0.0247	0.1121	0.1712	0.1384	0.0250
7	0.0461	0.1385	0.0947	0.0210	0.0457	0.1362	0.0941	0.0225
8	0.0326	0.1008	0.0772	0.0256	0.0320	0.0989	0.0774	0.0237
9	0.0244	0.0812	0.0563	0.0345	0.0240	0.0822	0.0559	0.0231
10	0.0185	0.0669	0.0391	0.0208	0.0182	0.0653	0.0387	0.0194
11	0.0142	0.0330	0.0214	0.0121	0.0145	0.0336	0.0218	0.0117
12	0.0108	0.0234	0.0186	0.0097	0.0106	0.0232	0.0184	0.0096
13	0.0083	0.0206	0.0164	0.0063	0.0081	0.0206	0.0163	0.0064

Tabla 4.17: R*esultados del algoritmo* SRBF *aplicado a la función pom descrita en la ecuación (4.18).*

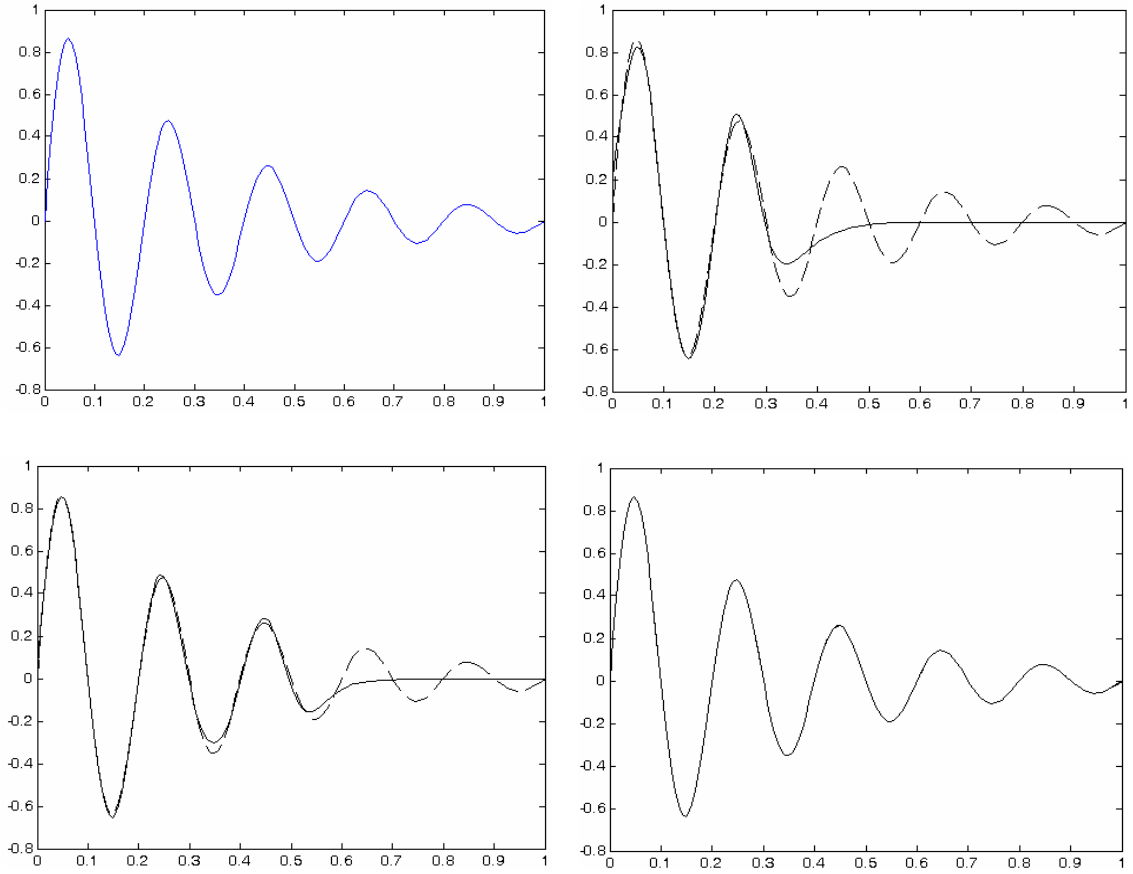


Figura 4.13: *Función pom descrita en la ecuación (4.18), y diferentes aproximaciones mediante redes de 3, 5 y 11 funciones base respectivamente.*

Algoritmo	m	n_p	NRMSE
[Pomares, 2000]	7	12	0.304
	8	14	0.237
	9	16	0.182
	10	18	0.153
	11	20	0.126
	12	22	0.114
[González, 2001]	4	12	0.2727 ± 0.0520
	5	15	0.1817 ± 0.0549
	6	18	0.1366 ± 0.0397
	7	21	0.0896 ± 0.0354
	8	24	0.0666 ± 0.0486
[Rivera, 2003]	4	12	0.3075 ± 0.0109
	5	15	0.2286 ± 0.0131
	6	18	0.1690 ± 0.0128
	7	21	0.1263 ± 0.0124
	8	24	0.1023 ± 0.0129
Algoritmo propuesto	4	12	0.2794 ± 0.0415
	5	15	0.1986 ± 0.0456
	6	18	0.1380 ± 0.0247
	7	21	0.0947 ± 0.0210
	8	24	0.0772 ± 0.0256

Tabla 4.18: *Comparativa de diferentes algoritmos en la aproximación de la función pom.*

4.2.3 Funciones de dos dimensiones

En este apartado estudiamos el comportamiento del algoritmo SRBF que aplicamos a la aproximación de diferentes funciones de dos dimensiones. Se presentaran tablas completas de resultados obtenidos tanto por el conjunto de entrenamiento como el de test, variando en cada ejecución el número de parámetros del sistema.

Al final de cada sección se presentara una tabla de los resultados más relevantes que obtenemos comparados a otros trabajos, que son referencia en este tipo de tareas como son los de Pomares [Pomares, 2000], González [González, 2001] y Rivera [Rivera, 2003].

Para cada función se ha trabajado con un conjunto de entrenamiento de 400 puntos distribuidos en una rejilla de 20 por 20 celdas, mientras el conjunto de vectores de test ha sido repartido en una rejilla de 31 por 31 celdas de forma aleatoria, formando un conjunto de 961 puntos. Estos conjuntos de entrenamiento y test son similares a los utilizados en [Pomares, 2000], [González, 2001], [Rivera, 2003] y a otros trabajos como el de [Cherkassky, 1996] que utiliza el mismo número de puntos, pero distribuidos en espiral.

4.2.3.1 Función y_5

Esta función ha sido propuesta por Rovatti y Guerrieri [Rovatti & Guerrieri, 1996] para comprobar el funcionamiento de un algoritmo de construcción de sistemas difusos en el que tanto las funciones de pertenencia en la entrada como en la salida estaban fijas a priori, con lo que simplemente se realizaba un proceso de minimización simbólica.

Matemáticamente, esta función se escribe de la forma siguiente:

$$y_5(x_1, x_2) = \frac{1}{2}\left[1 + \sin(2\pi x_1)\cos(2\pi x_2)\right] \qquad x_1, x_2 \in [0,1] \tag{4.19}$$

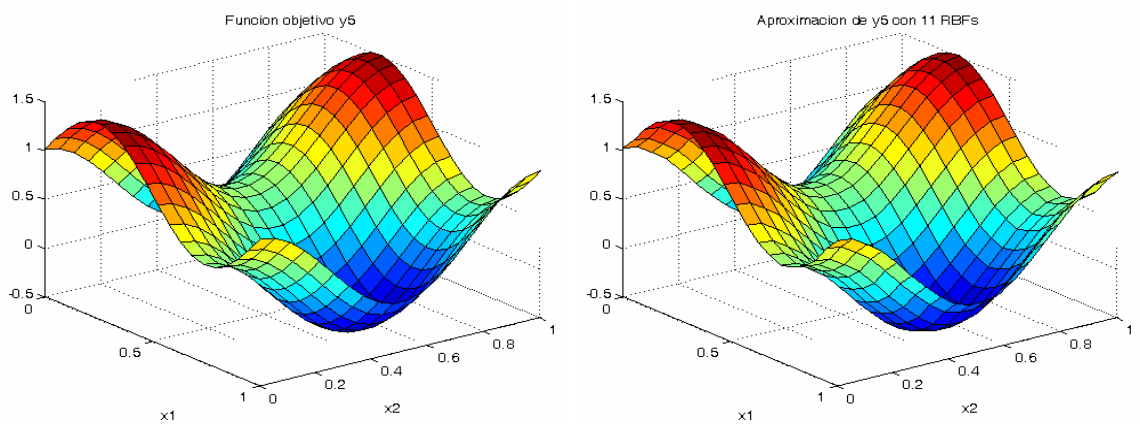


Figura 4.14: *Función y5 descrita en la ecuación (4.19) y su aproximación óptima.*

En [Rovatti & Guerrieri, 1996], tanto el número de reglas como el número de funciones de pertenencia se fijan de antemano antes de ejecutar el algoritmo. Rojas en [Rojas, 2000] presentó el método SOFRG, un algoritmo de identificación de sistemas difusos a partir de un conjunto de ejemplos que también optimizaba el número de reglas mediante el uso de un índice que determinaba la controversia entre las mismas. Posteriormente, Pomares [Pomares et al, 2000] también utilizó la función *y5* para comparar los resultados obtenidos por su algoritmo de identificación sistemática de sistemas difusos.

La tabla de resultados de la función *y5* 4.19 muestra un comportamiento coherente de algoritmo SRBF, el error de aproximación disminuye conforme se va añadiendo RBFs, mientras los valores de las desviaciones estándar testifican de la robustez de nuestro algoritmo.

La tabla 4.20 representa una comparativa de los resultados de nuestro algoritmo y los resultados obtenidos por los trabajos de referencia en esta tarea, tanto los basados en sistemas de inferencia difusa como los que recurren a redes neuronales de funciones RBF. Los valores obtenidos sitúan nuestro algoritmo como la mejor propuesta para la aproximación de esta función, los valores del error para cada configuración de parámetros son óptimos, lo que confirma el buen comportamiento que viene siendo probado a lo largo de este capitulo.

RBFs	*Entrenamiento NRMSE*				*Test NRMSE*			
	Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
3	0.3068	0.3075	0.3072	0.0002	0.3066	0.3075	0.3074	0.0003
4	0.1722	0.1975	0.1820	0.0076	0.1715	0.1968	0.1818	0.0068
5	0.0607	0.1568	0.1038	0.0321	0.0596	0.1554	0.1014	0.0315
6	0.0322	0.0760	0.0534	0.0172	0.0335	0.0762	0.0541	0.0184
7	0.0187	0.0355	0.0271	0.0084	0.0178	0.0353	0.0266	0.0087
8	0.0095	0.0207	0.0156	0.0036	0.0095	0.0194	0.0150	0.0034
9	0.0051	0.0102	0.0078	0.0020	0.0049	0.0106	0.0080	0.0023
10	0.0018	0.0058	0.0040	0.0012	0.0020	0.0063	0.0042	0.0013
11	0.0008	0.0020	0.0015	4.1E-4	0.0008	0.0021	0.0015	4.2E-4
12	0.0008	0.0015	0.0012	2.1E-4	0.0008	0.0015	0.0012	2.1E-4

Tabla 4.19: R*esultados del algoritmo* SRBF *aplicado a la función y5 descrita en la ecuación (4.19).*

Algoritmo	m	n_p	RMSE	NRMSE
[Rovatti, 1996]	5×5	25	0.130	-
[Rojas, 2000]	4×5	25	0.142	-
	6×7	51	0.068	-
	7×8	67	0.052	-
[Pomares, 2000]	4×4	20	0.080	0.158
	5×5	31	0.030	0.122
	6×6	44	0.014	0.056
[González, 2001]	5	20	0.0643 ± 0.0114	0.2556 ± 0.0454
	6	24	0.0484 ± 0.0094	0.1925 ± 0.0372
	8	32	0.0252 ± 0.0061	0.1003 ± 0.0242
	10	40	0.0131 ± 0.0036	0.0522 ± 0.0143
	11	44	0.0109 ± 0.0058	0.0433 ± 0.0232
[Rivera, 2003]	5	20	0.0673 ± 0.0072	0.2676 ± 0.0331
	6	24	0.0479 ± 0.0111	0.1905 ± 0.0440
	8	32	0.0231 ± 0.0020	0.0915 ± 0.0079
	10	40	0.0066 ± 0.0012	0.0264 ± 0.0045
	11	44	0.0056 ± 0.0011	0.0222 ± 0.0045
Algoritmo propuesto	5	20	0.0264 ± 0.0081	0.1038 ± 0.0321
	6	24	0.0135 ± 0.0043	0.0534 ± 0.0172
	8	32	0.0039 ± 0.0009	0.0156 ± 0.0036
	10	40	0.0011 ± 0.0003	0.0040 ± 0.0012
	11	44	3.7E-4 ± 1.0E-4	$0.0015 \pm$ 4.1E-4

Tabla 4.20: *Comparación de los resultados obtenidos para la función y5 por el algoritmo SRBF y otros algoritmos de aproximación de funciones.*

4.2.3.2 Función f_6

La función objetivo *f6* fue originalmente utilizada en el algoritmo de Cherkassky [Cherkassky & Lari-Najafi, 1991] donde presentaba una comparativa de distintos paradigmas utilizaos para la aproximación de funciones. Entre los métodos comparados se encuentran el *Projection Pursuit* (PP) [Friedman, 1981], un método adaptativa que utiliza sumas de funciones que dependen linealmente de las entradas, el método de *Multivariate adaptive regression splines* (MARS) [Friedman, 1991], que de manera adaptativa divide el espacio de entrada en regiones inconexas y modela cada región con un valor constante, el *Constrained topological mapping* (CTM) [Cherkassky & Lari-Najafi, 1991] muy parecido a MARS donde se usan mapas auto-organizativas para determinar la partición inicial, y un Perceptrón multicapa (MLP) con 15 neuronas. Esta función se escribe de la menara siguiente:

$$f\left(x_1, x_2\right) = 1.3356 \cdot \left[\begin{array}{l} 1.5\left(1-x_1\right) + e^{2x_1-1}\sin\left(3\pi\left(x_1-0.6\right)^2\right) + \\ +e^{3\left(x_2-0.5\right)}\sin\left(4\pi\left(x_2-0.9\right)^2\right) \end{array}\right] \qquad (4.20)$$
$$x_1, x_2 \in [0,1]$$

En [Cherkassky et al, 1996], además de los paradigmas anteriores, se introducen métodos bastante mejorados para la optimización de perceptrones multicapa, logrando resultados realmente notables. Los métodos basados en redes neuronales presentaban 40 neuronas en la capa oculta. Esto implica un número total de 161 parámetros (40 por 2, otros 40 pesos entre las neuronas ocultas y la salida, más (40+1) *bias*). En [Pomares et al, 2000] se utiliza un algoritmo para la identificación de sistemas difuso con un número similar de parámetros libres para aproximar dicha función. Esta función objetivo ha sido también utilizada en [Castillo, 2000] y [Castillo et al, 2001] para comprobar los resultados obtenidos por G-Prop, un algoritmo que evoluciona perceptrones multicapa.

La tabla 4.21 representa los resultados de la aproximación de esta función mediante el algoritmo que proponemos en esta memoria, se consigue una disminución paulatina del error conforme se van añadiendo RBFs al sistema.

Comparando los resultados experimentales obtenidos para esta función con los trabajos previamente descritos, consideramos que son realmente óptimos, el número de parámetros que hemos necesitado para la aproximación de esta función es sensiblemente inferior a los demás trabajos, la explicación que podemos avanzar para esta buena aproximación es la forma grafica de la función objetivo, como muestra la representación bidimensional, esta función es al mismo tiempo muy inconstante respecto al eje de ordenadas mientras sigue una variación previsible si se estudia del punto de vista de abscisas y se pueden exprimir como la pre-ecuación siguiente:

$$f_6(x_1, x_2) - f_6(x_1, x'_2) \approx f_6(x'_1, x_2)\, f_6(x'_1, x'_2) \qquad \forall (x_1, x'_1, x_2, x'_2) \in [0,1] \tag{4.21}$$

Esta configuración permite que la aproximación de la función *f6* mediante el algoritmo SRBF ya que cada RBF, además de tener el efecto local convencional alcanzando un valor máximo en el centro, también influye en el sentido de la proyección de este centro sobre cada eje de los vectores de entrada como demuestran las figuras 3.2 y 3.5 del capítulo anterior.

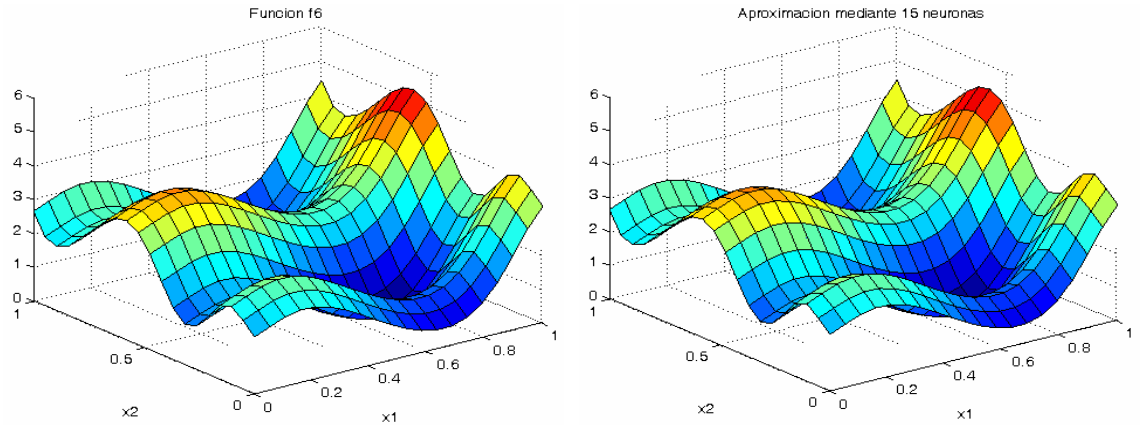


Figura 4.15: *Función f6 descrita en la ecuación (4.20) y su aproximación óptima.*

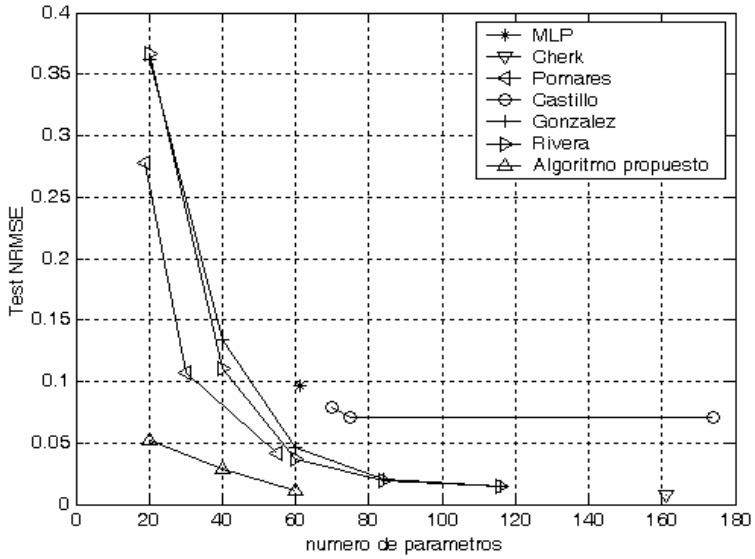


Figura 4.16: *Grafico comparativo de los resultados obtenidos por SRBF y por otros trabajos de referencia.*

RBFs	*Entrenamiento NRMSE*				*Test NRMSE*			
	Mejor	*Peor*	*Medio*	*Desv.*	*Mejor*	*Peor*	*Medio*	*Desv.*
5	0.0420	0.0684	0.0523	0.0112	0.0417	0.0682	0.0520	0.0120
6	0.0384	0.0581	0.0494	0.0087	0.0389	0.0582	0.0498	0.0090
7	0.0359	0.0482	0.0437	0.0072	0.0362	0.0471	0.0430	0.0069
8	0.0311	0.0437	0.0380	0.0074	0.0308	0.0428	0.0375	0.0076
9	0.0271	0.0379	0.0321	0.0051	0.0284	0.0394	0.0328	0.0068
10	0.0206	0.0340	0.0284	0.0085	0.0217	0.0347	0.0290	0.0082
11	0.0158	0.0337	0.0254	0.0060	0.0153	0.0310	0.0238	0.0054
12	0.0137	0.0309	0.0205	0.0092	0.0135	0.0302	0.0191	0.0081
13	0.102	0.0258	0.0166	0.0087	0.0098	00258	0.0164	0.0091
14	0.0068	0.0194	0.0130	0.0058	0.0068	0.0187	0.0127	0.0057
15	0.0042	0.0170	0.0106	0.0053	0.0041	0.0168	0.0102	0.0053

Tabla 4.21: R*esultados del algoritmo SRBF aplicado a la función f6 descrita en la ecuación*(4.20).

Algoritmo	m	n_p	Test NRMSE
MLP [Cherkassky, 1991]	15	61	0.096
PP [Friedman, 1981]	-	-	0.128
CTM [Cherkassky, 1991]	-	-	0.170
MARS PP [Friedman, 1991]	-	-	0.063
[Cherkassky et al, 1996]	40	161	0.008
[Pomares, 2000]	3×5 (PT)	19	0.278
	4×6 (PT)	30	0.104
	5×9 (PT)	55	0.041
[Castillo, 2000]	G-Prop (fn)	174 ± 33	0.071 ± 0.006
	G-Prop (fl)	70 ± 37	0.079 ± 0.008
	G-Prop (fd)	75 ± 14	0.071 ± 0.006
[González, 2001]	5	20	0.3622 ± 0.0268
	10	40	0.1343 ± 0.0261
	15	60	0.0459 ± 0.0096
	21	84	0.0200 ± 0.0054
	29	116	0.0140 ± 0.0045
[Rivera, 2003]	5	20	0.3666 ± 0.0168
	10	40	0.1108 ± 0.0135
	15	60	0.0368 ± 0.0092
	21	84	0.0191 ± 0.0036
	29	116	0.0147 ± 0.0022
Algoritmo propuesto	5	20	0.0523 ± 0.0112
	10	40	0.0284 ± 0.0085
	15	60	0.0106 ± 0.0053

Tabla 4.22: *Comparativa de los resultados obtenidos por los trabajos de referencia y nuestro algoritmo aplicados a la función f6.*

4.3 Conclusiones

En este capitulo se han presentado los resultados generados por el algoritmo SRBF que se propone en este trabajo como método bio-inspirado para la aproximación de funciones y en general como método de optimización, tras crear un conjunto de funciones sintéticas de una y dos dimensiones para realizar simulaciones comparativas, se ha escogido una batería de funciones objetivos de una y dos dimensiones ampliamente utilizadas en la biografía para esta misma tarea de aproximación de funciones. Se ha establecido un conjunto de tablas de resultados donde se aprecia la evolución del algoritmo que proponemos conforme va aumentando la complejidad del sistema, así como otro tipo de tablas comparativas con los trabajos de referencia antes mencionados, estos trabajos basados en técnicas bio-inspiradas como sistemas de inferencia difusa, perceptrones multicapa y redes neuronales artificiales parten de un esquema pre-establecido de condiciones iniciales que se ha respetado al realizar las ejecuciones, así pues, tanto en las funciones de una dimensión como en las bidimensionales, los conjuntos de datos de entrenamiento como los de test se han establecido de manera similar a los ya establecidos en la bibliografía.

A cada función utilizada en este capítulo, precede un breve comentario del origen de su aplicación y los diferentes trabajos que la mencionan. Del mismo modo, y para cada función, se ha establecido una tabla comparativa de resultados. Ordenados por orden cronológico, estos resultados se presentan en función del número de parámetros utilizados en cada experimento, para reflejar fielmente el resultado final.

Cuando se ha considerado relevante, se ha realizado figuras representativas adicionales, como pueden ser las comparativas entre funciones originales y puntos de salida generados por el algoritmo SRBF en su configuración óptima, así como gráficos comparativos de los resultados de diferentes algoritmos respecto al número de parámetros utilizados.

A la luz de estas condiciones, se pueden sacar las conclusiones siguientes:

- Respecto a la funciones sintéticas propuestas en este capítulo, se han establecido a partir de una formula sencilla para tener la posibilidad de realizar una

- comparación entre los valores de los parámetros obtenidos en las simulaciones y los valores reales de estas funciones.
- Las tablas comparativas muestran una gran destreza del algoritmo propuesto, los valores obtenidos aplicando este algoritmo mejoran sustancialmente los obtenidos mediante un algoritmo evolutivo convencional, este hecho se aprecia en todas y cada una de las funciones utilizadas.
- El escalonamiento de los valores del error muestran la coherencia de nuestro algoritmo, efectivamente, en todas las funciones utilizadas los valores del NRMSE van disminuyendo conforme se aumenta la complejidad del sistema.
- Los resultados obtenidos a partir de los conjuntos de entrenamiento y los obtenidos a partir de los conjuntos de test son bastante parejos, llegando a igualarse en algunos casos, este dato es de una importancia primordial en un algoritmo de estas características y testifica de una buena generalización.
- Los valores de las desviaciones estándar son relativamente reducidos. De cada experimento se han realizado 10 ejecuciones, y se calculan la media de estos valores y la desviación estándar que testifica de cuanto de agrupados están estos valores. Estos valores a lo largo de este capitulo muestran una gran robustez del algoritmo propuesto.
- El elemento más representativo de este capitulo son las tablas comparativas de las diferentes técnicas de aproximación de funciones. El algoritmo SRBF genera unos resultados que mejoran de forma rotunda los algoritmos basados en sistemas de inferencia difusa, mientras que comparado a otros algoritmos de referencia basados en redes neuronales, exceptuando algunas situaciones aisladas, nuestro algoritmo presenta unas mejorías sustanciales en el resto de los casos, con la calidad destacable como es el bajo coste computacional que supone la implantación del algoritmo SRBF.

Capítulo 5/
Conclusiones y aportaciones

A continuación se sintetizan las conclusiones extraídas de este trabajo, así como las principales líneas de investigación que se abren a la luz de las implementaciones realizadas. Este trabajo de tesis doctoral se sitúa en la línea de los sistemas basados en técnicas bio-inspiradas orientadas a tareas de optimización, centrándose en el problema de aproximación de funciones, mediante redes neuronales de base radial.

En este trabajo se desarrolla una nueva estructura de red neuronal tipo RBF, centrándonos en nuevas metodologías para la realización de operaciones algebraicas en el cómputo de la activación de cada RBF. La idea principal que ha inspirado ésta tesis es forzar una implicación más relevante de operadores tales como T-conorma en la construcción de las neuronas de la capa oculta en un sistema RBF. Esta tesis es una continuación de varios trabajos anteriores, tanto los que consideramos de referencia, realizados en este mismo departamento como otros trabajos que vienen mencionados en los capítulos anteriores enfocados a los problemas de optimización en general y al de la aproximación de funciones en particular. La implementación de las funciones de base radial gaussianas en la composición de la redes neuronales artificiales proporciona unas interpolaciones que permiten una reducción sustancial en los niveles de error obtenidos.

En Esta tesis se propone un nuevo algoritmo denominado S-RBF, en el que se incorporan procedimientos evolutivos, con otros algoritmos de optimización, para el aprendizaje de una nueva estructura de red neuronal con funciones de base radial.

Como paso previo a la fase del proceso evolutivo, se utiliza un algoritmo de clustering de datos, quedando demostrado que una buena inicialización de parámetros contribuye a un desarrollo menos costoso de un algoritmo de aproximación de funciones sin perder de su aptitud de generalización.

A esta novedosa orientación de las redes neuronales artificiales se añade un factor propio a cada conexión entrada-neurona, y que pondera la influencia de cada función base en la aproximación de un vector de entrada dado, permitiendo del mismo modo privilegiar las funciones con más aporte a nuestra red, y descartar aquellas funciones menos efectivas.

Esta estrategia se ve mejorada por los aportes de heurísticas de determinación de los pesos de la red, esta fase del algoritmo ha sido dividida en dos partes, la primera, basada en mínimos cuadrados, acompaña la evolución de la solución dentro del proceso genético mientras que al final del proceso de optimización se realiza una descomposición en valores singulares que permite obtener los pesos más óptimos para la configuración final.

En la fase experimental de esta tesis, se aplica el algoritmo SRBF a una batería de funciones de una y dos dimensiones que han sido utilizadas en trabajos anteriores para estudiar otros algoritmos orientados a la misma tarea, los resultados obtenidos reflejan las siguientes características:

- Aplicando una serie de ejecuciones a una misma función pero con configuraciones cada vez más complejas, muestra unas evoluciones coherentes de los niveles del error, el algoritmo que proponemos no encuentra problemas de sobre-entrenamiento.
- La evolución de los valores de aptitud muestran un desarrollo equilibrado, los operadores genéticos implantados evitan caer en óptimos locales que en otros casos provocan una convergencia prematura y frenan el proceso de optimización.

- Los niveles de la desviación estándar a lo largo de los diferentes experimentos son menores, y testifican de una buena generalización del sistema implantado.
- Respecto a las funciones sintéticas introducidas en este trabajo, una exhaustiva comparación entre la aproximación de estas funciones mediante un algoritmo genético convencional y el algoritmo que proponemos, testifica de la robustez y la fiabilidad de nuestro sistema.
- Gracias por una parte al establecimiento de una condición de parada adecuada al tipo de problemas tratados, y por otra a la estrategia seguida en la construcción de nuestra red neuronal, se consigue evitar el prolongamiento innecesario de una ejecución, y un ahorro determinante en el coste computacional de la implantación.
- Las tablas de comparaciones entre los resultados obtenidos por diferentes métodos bio-inspirados para la aproximación de funciones considerados como referencia muestran las mejorías que aporta nuestro algoritmo, tanto en los niveles del error como en complejidad del sistema.
- La introducción del factor de ponderación actuando en cada conexión entrada-neurona, tiene un papel primordial en el algoritmo SRBF, regulariza, a través del proceso genético, la implicación de cada función base en la aproximación de un vector de datos propiciando de este modo el hipotético mantenimiento o reemplazo de cada una de estas funciones.

En cuanto a las futuras líneas de investigación, se presentan dos opciones principales:

- Dado que la construcción de este algoritmo se basa sobre una novedosa orientación de la manera de activación de las RBFs dentro de una red neuronal, nos fijamos como objetivo principal seguir en esta línea, desarrollando algoritmos con componentes algébricas capaces de adaptarse a los problemas de optimización que tratamos.
- Consideramos la construcción de un algoritmo que combina a la vez la estructura de S-RBF con otras estructuras más convencionales y hacer que,

de forma autónoma, el algoritmo escoja la “vía” más adecuada para el problema que se le presenta.

- También se ha implantado otra novedosa estructura de RBF, basada en la idea anterior de utilización de nuevos operadores para la activación de las neuronas , pero en este caso se utilizara una salida. A diferencia de la salida convencional de una RBF consistente en un solo peso, ahora se propone que la nueva estructura neuronal esté compuesta en la salida por una expresión polinomial de las variables de entrada, sopesadas por ciertos pesos o sinapsis neuronales.

Apéndice A/
Sistema S-RBF con regresión de la variable de salida

A.1 RBF Convencionales

Las redes de Funciones de Base Radial (o redes RBF) son un tipo especial de red neuronal con una topología y funcionamiento específico. Tanto la estructura como los algoritmos de aprendizaje que presentan les dan unas características y propiedades especiales, que fuerzan un estudio específico, aparte de las redes neuronales tradicionales como el Perceptrón Multicapa. Aparte, las redes de funciones de base radial forman parte de la base de paradigmas que actualmente están muy en auge como son las Máquinas de Vectores Soporte y los Procesos Gaussianos. Estas razones justifican el estudio realizado en este trabajo sobre diferentes modificaciones aplicables a las redes RBF para problemas de aproximación funcional.

Las redes RBF constan de tres capas: una capa interna de conexión de la red al entorno (a los valores de entrada, por esto a veces se considera que solo poseen dos capas reales); una segunda capa o capa oculta cuyas neuronas llevan asociadas un centro y una varianza de dimensiones iguales a los datos de entrada de la red, y que aplican alguna función de base radial obteniendo una salida para cada dado de entrada; y finalmente una tercera capa o capa de salida que obtiene una salida (o varias salidas) que es función lineal de las salidas obtenidas por las neuronas en la capa oculta a partir de los datos de entrada, según los pesos de la red $w_i^{(output)}$ (ver figura A.1).

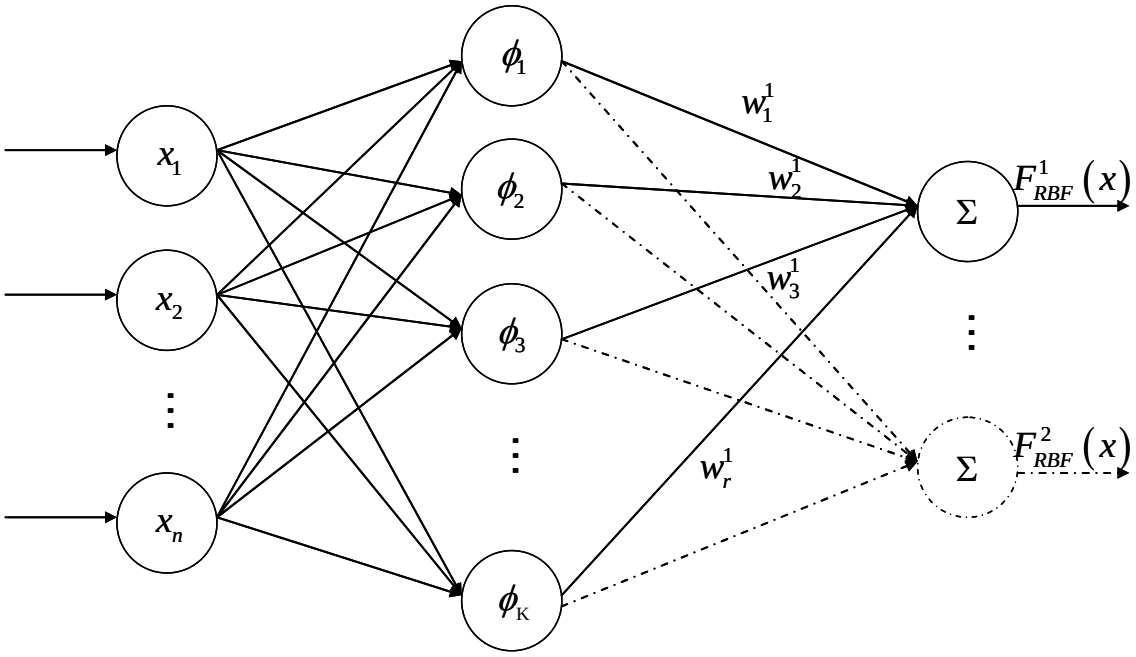


Figura A.1: *Estructura de una red neuronal de funciones de base radial.*

Así, en general la capa de salida funciona como una unidad de suma, de forma que la salida de la red RBF se expresa como

$$F_{RBF}(\vec{x}^m) = \sum_{i=1}^{K} w_i \phi_i(\vec{x}^m, c_i, \sigma_i) \tag{A.1}$$

Las funciones de base radial $\phi_i(\vec{x}_n, c_i, \sigma_i)$ son funciones no lineales

$$\phi_i\left(\vec{x}^m, c_i, \sigma_i\right) = \phi_i\left(\frac{\left\|\vec{x}^m - c_i\right\|}{\sigma_i}\right) \tag{A.2}$$

donde la función ϕ es normalmente una gaussiana, aunque puede utilizarse cualquier tipo de función de base radial (multi-cuadráticas, cúbicas, …)

$$\phi(r) = \exp\left(\frac{r^2}{2}\right) \tag{A.3}$$

y donde $\|\cdot\|$ es la norma euclídea que se define como

$$\left\|\vec{x}^m - c_i\right\| = \left(\sum_{j=1}^{n}\left(x_j^m - c_{ij}\right)^2\right)^{\frac{1}{2}} \tag{A.4}$$

donde *r* es el radio o la distancia calculada como se indica en (A.2) y (A.4).

Así, la función ϕ puede expresarse como

$$\phi_i\left(\vec{x}^m, c_i, \sigma_i\right) = \prod_{j=1}^{n} \exp\left(-\frac{\left(c_{ij} - x_j^m\right)^2}{\sigma_{ij}^2}\right) \tag{A.5}$$

Pudiendo ser la anchura de la función de base radial σ_{ij}^2 distinta para cada dimensión del espacio de entrada, o igual para todas las dimensiones (simplemente σ_i^2 para la RBF *i*).

Una alternativa en la expresión lineal de la salida de la red para la capa de salida es calcular la suma ponderada de las funciones de base radial existentes en la red. En este tipo de redes por tanto la salida queda normalizada por el total de las activaciones de la capa oculta. La expresión de la salida para la red $F_{RBF}^*(\vec{x}^m)$ queda expresada de la siguiente forma

$$F_{RBF}^*(\vec{x}^m) = \frac{\sum_{i=1}^{K} w_i \phi_i(\vec{x}^m, c_i, \sigma_i)}{\sum_{i=1}^{K} \phi_i(\vec{x}^m, c_i, \sigma_i)} \tag{A.6}$$

A.2 Utilización de pesos de regresión en la red neuronal de funciones de base radial

En las funciones de base radial, los parámetros que conectan las activaciones de las neuronas ocultas con la neurona de salida w_i, normalmente son parámetros sencillos (constantes). Sin embargo existe la posibilidad de generalizar la expresión de dichos parámetros, aumentando por tanto su capacidad expresiva, utilizando lo que se llaman pesos de regresión ("regression weights"). Así, los pesos de la red RBF w_i, pasan a ser función de las variables de entrada, mediante la adición de conexiones laterales entre las neuronas de funciones de base radial

$$w_i = h(x_1, ..., x_n) \tag{A.7}$$

Siendo típicamente esta función $h(\ )$ simplemente una función lineal de las variables de entrada, según los parámetros $b_{i1}, ..., b_{in}$ para la RBF ϕ_i (ver figura A.2)

$$w_i = \sum_{j=1}^{n} b_{ij} x_j + b_{i0} \tag{A.8}$$

Utilizando esta formulación, aumenta el número de parámetros de la red RBF con lo que el proceso de optimización de la red (así como el proceso de evaluación) aumentan su complejidad. Sin embargo, este tipo de parámetros tienen una optimización relativamente simple, pues se pueden obtener mediante aprendizaje supervisado utilizando optimización lineal, como veremos en la sección de optimización de redes RBF. Así, la capacidad aproximadora de cada neurona en la capa oculta aumenta, y el número de neuronas necesarias para realizar la aproximación funcional se puede ver reducido.

Esta última formulación (A.7) y (A.8), puede ser ampliada para considerar una expresión polinómica de orden mayor, por ejemplo de orden 2, con lo que la expresión de los pesos quedaría de la siguiente forma

$$w_i = b_{i0} + \vec{b}_{i1} \cdot \vec{x} + \vec{x}^T B_{i2} \vec{x} \tag{A.9}$$

Donde b_{i0} es el coeficiente de orden cero, $\vec{b}_{i1}$ es un vector con los coeficientes de primer orden, y B_{i2} es una matriz triangular superior con los coeficientes de segundo orden para la RBF ϕ_i.

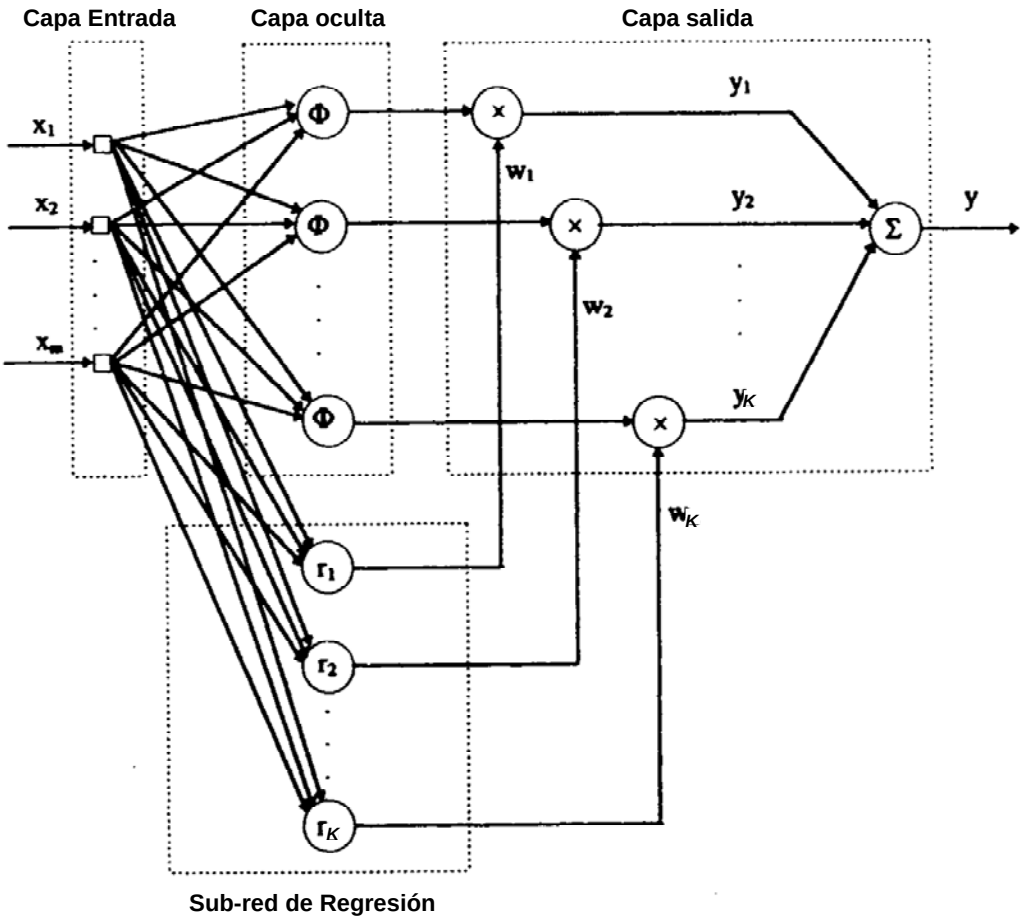


Figura A.2. *Estructura de una red neuronal de base radial con pesos de regresión*

La figura A.2 muestra la estructura de una red de funciones de base radial con coeficientes de regresión. Las neuronas añadidas mediante conexiones laterales $r_1,...,r_K$ en forma de sub-red de regresión, realizan el cómputo de los pesos $w_1,...,w_K$. La capa de salida en la práctica queda dividida en dos sub-capas, una para la aplicación de los pesos a las salidas de las neuronas en la capa oculta y otra que aplica la función lineal sumatoria de salida.

A.3 Aplicación de diferentes operadores T-norma a la formulación de las funciones de base radial

De (A.1) y (A.5), obtenemos la expresión general completa de la salida de una red neuronal en función de todos sus parámetros:

$$F_{RBF}(x^m) = \sum_{i=1}^{K} w_i \phi_i(x^m, c_i, \sigma_i) = \sum_{i=1}^{K} w_i \prod_{j=1}^{n} \exp\left(-\frac{\left(c_{ij} - x_j^m\right)^2}{\sigma_{ij}^2} \right) \qquad \text{(A.10)}$$

Vista esta expresión general podrían aplicarse ciertas modificaciones, sin llegar a romper la estructura fundamental de la red RBF. Así, esta última formulación puede modificarse de forma que cada RBF *n*-dimensional pase a ser un conjunto de *n* RBFs, cada una de una sola dimensión, de forma que la salida de la red RBF quedaría de la siguiente forma

$$F_{RBF}(x^m) = \sum_{i=1}^{K} w_i \sum_{j=1}^{n} \phi_{ij}(x^m, c_{ij}, \sigma_{ij}) = \sum_{i=1}^{K} w_i \sum_{j=1}^{n} \exp\left(-\frac{\left(c_{ij} - x_j^m\right)^2}{\sigma_{ij}^2} \right) \qquad \text{(A.11)}$$

Y utilizando pesos para todas las neuronas por separado

$$F_{RBF}(x^m) = \sum_{i=1}^{K}\sum_{j=1}^{n} w_i \phi_{ij}(x^m, c_{ij}, \sigma_{ij}) = \sum_{i=1}^{K}\sum_{j=1}^{n} w_i \exp\left(-\frac{\left(c_{ij} - x_j^m\right)^2}{\sigma_{ij}^2}\right) \qquad \text{(A.12)}$$

Sin embargo, la capacidad aproximadora de estas redes queda menguada considerablemente, al no involucrar ningún tipo de operación de tipo T-Norma en las mismas. Otra modificación posible a este tipo de red por tanto es la de modificar el producto (en el cálculo de la salida de una neurona) por otro tipo de operador de T-Norma. Esta formulación, flexibiliza la forma (A.10) y ofrece la posibilidad de añadir nuevas propiedades a la red RBF. En este trabajo vamos a realizar un pequeño estudio de la repercusión de la introducción de varios de estos operadores al funcionamiento de una red RBF.

Primeramente presentamos un operador parametrizado que combina las expresiones (A.10) y (A.11), de manera que se pueda valorar tanto la suma de todas las activaciones en todas las dimensiones como el producto de dichas activaciones. La expresión resultante para la activación de una red neuronal tomando este operador que llamaremos de Ponderador_Suma

$$\phi_i^O\left(x^m, c_i, \sigma_i\right) = \lambda \cdot \prod_{j=1}^{n} \exp\left(-\frac{c_{ij} - x_j^m}{\sigma_{ij}^2}\right) + (1-\lambda)\cdot \sum_{j=1}^{n} \exp\left(-\frac{\left(c_{ij} - x_j^m\right)^2}{\sigma_{ij}^2}\right) \qquad \lambda \in [0,1]$$

(A.13)

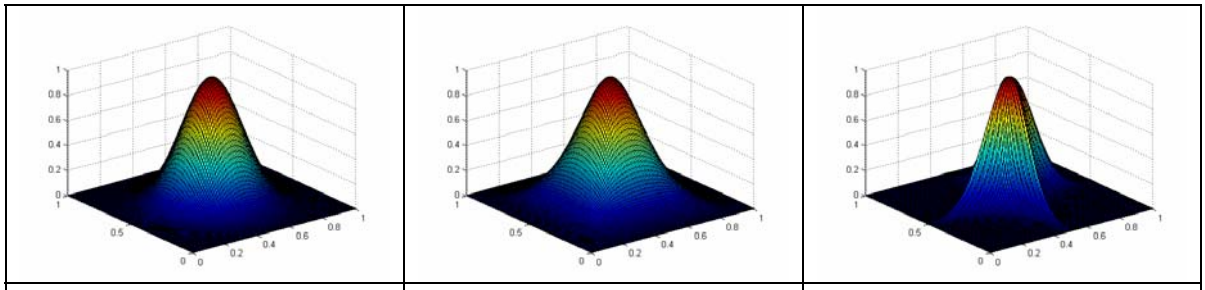
El parámetro λ controla la importancia de un factor respecto de otro en la activación de la neurona ϕ_i. Como posibles valores para este parámetro se han tomado $\lambda = 0.01$, $\lambda = 0.5$ y $\lambda = 1$ (nótese que para $\lambda = 1$ la ecuación queda reducida a (A.13). Este operador carece de ciertas propiedades necesarias para ser considerado operador T-norma, como es la propiedad asociativa.

En segundo lugar presentamos el operador Hamacher, también parametrizable que para dos valores se expresa como

$$Hamacher(a,b) = \frac{\lambda \cdot a \cdot b}{1-(1-\lambda)\cdot(a+b-a\cdot b)} \qquad (A.14)$$

Este operador es bien conocido y existen varios estudios sobre su utilización en comparación con otros posibles operadores T-norma, en concreto para sistemas difusos. En este caso, el parámetro λ puede tomar cualquier valor mayor que cero. Los valores para los que se han realizado comparaciones son $\lambda = 0.01$, $\lambda = 0.5$, $\lambda = 1$ y $\lambda = 4$. Nótese que aquí igualmente para $\lambda = 1$ el operador queda simplificado al producto (A.5)

La figura A.3 muestran el resultado de operar dos variables gaussianas de centro 0.5 y varianza 0.15 en el intervalo [0,1], mediante el operador Hamacher con diferentes valores para el parámetro λ. Obsérvese la forma de la salida en comparación con el producto a) para valores extremos de λ b) y c).



a) $\lambda = 1$	b) $\lambda = 100$	c) $\lambda = 0.01$

Figura A.3: *Forma del resultado de operar dos variables gaussianas con el operador Hamacher para distintos valores del parámetro* λ .

A.4 Optimización de la red de funciones de base radial

Suponiendo que disponemos de un conjunto de datos $D = \left(x^m, y^m\right)$, $m = 1,2,...M$ que deben ser aproximados, el problema de entrenar la red neuronal de funciones de base radial puede ser formulado como un problema de optimización, donde la función de

error (por ejemplo el error cuadrático medio normalizado NRMSE) entre la salida real y la salida producida por la red tiene que ser minimizada con respecto a la estructura de la red. Esto es, se trata de minimizar el error obtenido en la aproximación con respecto al número de neuronas en la capa oculta, y minimizada con respecto a los parámetros de la red, que son los centros y sigmas de las neuronas ocultas y los pesos de salida de la red.

La fórmula del error cuadrático medio normalizado es la siguiente

$$NRMSE \equiv \sqrt{\frac{\overline{e^2}}{\sigma_z^2}} \tag{A.15}$$

Donde σ_z^2 es la varianza de los datos de salida, y $\overline{e^2}$ es el error cuadrático medio entre la salida obtenida y la salida deseada

$$\overline{e^2} = \sum_{m \in D} \frac{\left(y^m - F_{RBF}(x^m)\right)^2}{M} \tag{A.16}$$

Existen numerosos algoritmos y metodologías de optimización de redes RBF. En este trabajo no nos vamos a introducir en la problemática de la obtención del número óptimo de neuronas en la capa oculta para resolver el problema de aproximación funcional. Primeramente presentamos la forma en la que vamos a inicializar la red RBF para realizar la aproximación funcional, utilizando el algoritmo de las k-medias para obtener los centros de las neuronas en la capa oculta. Posteriormente veremos la optimización de los parámetros no lineales centros y anchuras de las funciones de base radial de la red, la cual hace uso en cada iteración del proceso de optimización lineal de los pesos de la red.

A.4.1 Inicialización de la red de funciones de base radial

En este trabajo, consideramos que se realiza una estimación inicial del número de neuronas en la capa oculta. Así, conocido dicho número de neuronas, se realiza el

proceso de optimización de la red. Sin embargo, los parámetros no lineales del sistema, que son los centros y anchuras de las funciones de base radial, necesitan un valor inicial pseudos-optimo antes de proceder a la obtención de una configuración óptima local de los mismos. Así, el primer paso en la construcción de la red RBF es la obtención de los valores iniciales $c_i = \{c_{i1},...,c_{in}\}$ y $\sigma_i = \{\sigma_{i1},...,\sigma_{in}\}$ para cada ϕ_i.

Existen numerosos métodos en la literatura para la obtención de los valores óptimos iniciales para los centros y anchuras de las redes RBF. Nosotros hemos tomado el método de clustering de las *K*-medias utilizando los datos de de entrada de entrenamiento en el proceso (siendo *K* el número de funciones de base radial) para obtener una estimación de los centros iniciales óptimos $c_i = \{c_{i1},...,c_{in}\}$. Después las anchuras de las RBFs $\sigma_i = \{\sigma_{i1},...,\sigma_{in}\}$ se ajustan teniendo en cuenta la distancia de un centro al centro más cercano. Esta inicialización subóptima nos garantiza que el proceso posterior de optimización local de descenso en gradiente para la obtención de los centros y anchuras obtendrá unos valores para estos parámetros sino óptimos, si pseudos-óptimos.

A.4.2 Optimización de los centros y anchuras de las funciones de base radial en la capa oculta.

Una vez el sistema está inicializado dado un número inicial de neuronas ocultas, teniendo por tanto una configuración inicial de centros $c_i = \{c_{i1},...,c_{in}\}$ y anchuras $\sigma_i = \{\sigma_{i1},...,\sigma_{in}\}$ para cada función de base radial, realizamos un proceso de aprendizaje supervisado utilizando técnicas basadas en el descenso en gradiente. Este tipo de técnicas considera una superficie del error (A.16) como función de los parámetros para los que se desea hallar el óptimo, y realiza un proceso iterativo para acercarse a un mínimo en la superficie multidimensional del error. Utilizando las derivadas parciales de dicha función de error en función de cada parámetro a optimizar, las técnicas de descenso en gradiente encuentran un mínimo local del error, y los valores de los

parámetros (en nuestro caso centros $c_i = \{c_{i1},...,c_{in}\}$ y anchuras $\sigma_i = \{\sigma_{i1},...,\sigma_{in}\}$) para los que dicho mínimo ocurre.

En cada evaluación de los nuevos valores obtenidos de los parámetros no lineales, se realiza una evaluación del sistema con dichos parámetros. Dicha evaluación utiliza la optimización lineal de los pesos de la red (que veremos en la siguiente subsección) para obtener una estimación final del error para dicha configuración. Así, el proceso de descenso en gradiente obtiene un mínimo local en el error y los valores de los parámetros $c_i = \{c_{i1},...,c_{in}\}$ y $\sigma_i = \{\sigma_{i1},...,\sigma_{in}\}$ para cada RBF asociados a dicho mínimo dada la función de coste a minimizar (A.16).

Dentro de los posibles métodos de descenso en gradiente existentes en la literatura, el método utilizado es el método Levenberg-Marquardt, que es una modificación del método de Gauss-Newton. Éste ha sido escogido frente a otros métodos debido a la eficiencia en tiempo computacional en comparación con otros métodos y por su robustez.

A.4.3 Optimización de los pesos de la red de funciones de base radial

La optimización de los pesos, podría realizarse mediante la utilización de algún método de gradiente descendiente. Sin embargo, se puede aplicar un método directo de optimización lineal, que se resuelve algebraicamente, utilizando el conjunto de datos de entrenamiento $\left(x^m, y^m\right)$, $m = 1,2,...M$. La metodología aplicada es la Optimización por Mínimos Cuadrados (LSE), que pretenden minimizar la función de coste (A.16) que abreviada queda como

$$J = \sum_{m \in D} \left(y^m - F_{RBF}(x^m) \right)^2 \qquad \text{(A.17)}$$

donde y^m es la salida deseada para el punto x^m, siendo $F_{RBF}(x^m)$ la salida obtenida por la red RBF.

Suponiendo que conocemos los centros y anchuras de las funciones de base radial en las neuronas ocultas de la red RBF, la función de salida resulta lineal con respecto a los parámetros de los pesos, y por tanto se pueden aplicar numerosos métodos matemáticos sencillos para resolver el sistema de ecuaciones resultante. De entre estos métodos, se ha escogido la Descomposición en Valores Singulares (SVD), debido a sus eficiencia y a la capacidad de eliminar valores irrelevantes que evita redundancia y obtener soluciones inmanejables.

Ahora mostramos las derivadas parciales de la función de error (A.17) con respecto a cada coeficiente de los pesos de la red, ya sea constante, lineal o cuadrático (o incluso de orden superior). Suponemos que el resto de parámetros de la red, los centros c_{ij} y anchuras σ_{ij} de las funciones de base radial ϕ_i están fijados. La expresión genérica para cualquier coeficiente de orden cero (b_{i0}), de orden uno ($b_{i1_1},...,b_{i1_n}$) o de orden dos ($B_{i2_{11}},...,B_{i2_{nn}}$) (ver eq. (A.9)) teniendo en cuenta salida normalizada (A.6) es

$$\frac{\partial J}{\partial b_{is}} = 2\sum_{m\in D}\left(y^m - \frac{\sum_{k=1}^{K}\phi_k(\vec{x}^m)\left(b_{i0}+\vec{b}_{i1}\cdot\vec{x}^m+\left(\vec{x}^m\right)^T B_{i2}\vec{x}^m\right)}{\sum_{k=1}^{K}\phi_k(\vec{x}^m)}\right)\frac{\phi_j(\vec{x}^m)\cdot f_{w_s^j}(\vec{x}^m)}{\sum_{k=1}^{K}\phi_k(\vec{x}^m)} \qquad (A.18)$$

$$s=\{0,1_1,...1_n,2_{11},...,2_{nn}\},\ j=1..K$$

Siendo $f_{w_s^j}(\vec{x}^m)$ igual a 1 para el coeficiente de orden cero s = 0, x_j^m para los coeficientes de orden 1 s = 1_j y $x_l^m\cdot x_h^m$ para los coeficientes de orden dos s = 2_{lh}. Así las derivadas parciales llevan a los tres siguientes tipos de ecuaciones que están presentes en el sistema de ecuaciones lineales a optimizar

$$for\ s=0,\ \frac{\partial J}{\partial b_{i0}}=0\ \Rightarrow$$
$$\sum_{k=1}^{K}\sum_{m\in D}\frac{\phi_k(\vec{x}^m)\left(b_{i0}+\vec{b}_{i1}\cdot\vec{x}^m+\left(\vec{x}^m\right)^T B_{i2}\vec{x}^m\right)\cdot\phi_j(\vec{x}^m)}{\sum_{j=1}^{K}\phi_j(\vec{x}^m)}=\sum_{m\in D}\frac{y^m\cdot\phi_j(\vec{x}^m)}{\sum_{k=1}^{K}\phi_k(\vec{x}^m)} \tag{A.19}$$

$$for\ s=1_l,\ \frac{\partial J}{\partial b_{i1_l}}=0\ \Rightarrow$$
$$\sum_{k=1}^{K}\sum_{m\in D}\frac{\phi_k(\vec{x}^m)\left(b_{i0}+\vec{b}_{i1}\cdot\vec{x}^m+\left(\vec{x}^m\right)^T B_{i2}\vec{x}^m\right)\cdot\phi_j(\vec{x}^m)\cdot x_l^m}{\sum_{j=1}^{K}\phi_j(\vec{x}^m)}=\sum_{m\in D}\frac{y^m\cdot\phi_j(\vec{x}^m)\cdot x_l^m}{\sum_{k=1}^{K}\phi_k(\vec{x}^m)} \tag{A.20}$$

$$for\ s=2_{lh},\ \frac{\partial J}{\partial b_{2_{lh}}^j}=0\ \Rightarrow$$
$$\sum_{k=1}^{K}\sum_{m\in D}\frac{\phi_k(\vec{x}^m)\left(b_{i0}+\vec{b}_{i1}\cdot\vec{x}^m+\left(\vec{x}^m\right)^T B_{i2}\vec{x}^m\right)\cdot\phi_j(\vec{x}^m)\cdot x_l^m\cdot x_h^m}{\sum_{j=1}^{K}\phi_j(\vec{x}^m)}=\sum_{m\in D}\frac{y^m\cdot\phi_j(\vec{x}^m)\cdot x_l^m\cdot x_h^m}{\sum_{k=1}^{K}\phi_k(\vec{x}^m)} \tag{A.21}$$

El resultado será un sistema de $\left(1+n+n\wedge 2/2\right)$ ecuaciones con el mismo número de incógnitas que se resolverá como hemos indicado.

A.5 Simulaciones

Primeramente como ejemplo veremos la aplicación del trabajo expuesto a una función bien conocida tomada de Cherkassky et al (1996)

$$f_2=\exp\left(x_1\cdot\sin\left(\pi\cdot x_2\right)\right)\qquad X\ \ uniform\ in\left[-1,1\right] \tag{A.22}$$

Se han realizado varias ejecuciones considerando 400 puntos aleatoriamente distribuidos en el espacio de entrada. Tomando un número de neuronas igual a 5, y considerando salida normalizada de la red RBF respecto de las activaciones de todas las reglas, los resultados que obtenemos para los diferentes tipos de T-norma expuestos y pesos de regresión de orden cero, uno y dos, son los siguientes

TIPO DE T-NORMA	TIPO DE PESOS	NRMSE (media +std)	T_e
Ponderador_Suma $\lambda = 0.01$	*Constantes*	0.2365 + 0.0734	3.2 + 0.1
	Pesos de regresión orden 1	0.1198 + 0.0168	4.5 + 0.2
	Pesos de regresión orden 2	0.0151 + 0.0043	5.8 + 0.2
Ponderador_Suma $\lambda = 0.5$	*Constantes*	0.2213 + 0.0639	3.1 + 0.1
	Pesos de regresión orden 1	0.0952 + 0.0292	4.3 + 0.2
	Pesos de regresión orden 2	0.0185 + 0.0064	5.8 + 0.3
Hamacher $\lambda = 0.01$	*Constantes*	0.4564+ 0.0684	2.0 + 0.1
	Pesos de regresión orden 1	0.2722+ 0.0743	3.2 + 0.1
	Pesos de regresión orden 2	0.0291+ 0.0166	4.9 + 0.2
Hamacher $\lambda = 0.5$	*Constantes*	0.2373 + 0.1560	2.3 + 0.1
	Pesos de regresión orden 1	0.0980 + 0.0404	3.3 + 0.2
	Pesos de regresión orden 2	0.0190 + 0.0078	4.8 + 0.2
Hamacher $\lambda = 4$	*Constantes*	0.1914 + 0.0678	2.7 + 0.1
	Pesos de regresión orden 1	0.1051 + 0.0408	3.2 + 0.2
	Pesos de regresión orden 2	0.0110 + 0.0009	4.6 + 0.2
Hamacher $\lambda = 1$ **≡ Ponderador_Suma** $\lambda = 1$ **≡ Producto**	*Constantes*	0.1788 + 0.1566	2.3 + 0.1
	Pesos de regresión orden 1	0.1109 + 0.0279	3.3 + 0.2
	Pesos de regresión orden 2	0.0164 + 0.0079	4.6 + 0.3

Tabla A.1: *Resultados experimentales de la aplicación del sistema con regresión*

En primer lugar observamos la gran capacidad de aproximación que proporcionan los pesos de regresión de orden uno y aún más notable es la capacidad aproximadora de los pesos de segundo dos. Así, tomando como ejemplo la última fila de la tabla en la que consideramos el producto para el cálculo de las activaciones de las neuronas (A.5), para llegar a obtener el mismo error tomando pesos de regresión de orden 2, harían falta más de veinte neuronas ocultas, con un tiempo de ejecución en la optimización cinco veces superior.

Por otro lado observamos que comparando los diferentes operadores T-norma parametrizados utilizados, obtenemos resultados comparables con el operador tradicional producto. En este ejemplo utilizando el operador Hamacher con $\lambda = 4$ se obtienen los mejores resultados. Recordemos que conforme el parámetro λ crece por encima del valor 1, la activación de la neurona se suaviza conforme alguno de los valores operandos es mayor, aunque con una fuerte dependencia del resto de valores operandos.

En cuanto a la normalización de las activaciones de las neuronas ocultas, los resultados obtenidos normalizando las activaciones (A.6) (resultados mostrados en esta tabla) son mejores que simplemente tomando la neurona de salida como una unidad de suma (A.1) , lo que confirma estudios realizados al respecto en la literatura.

Bibliografía

[Amari, 1990] S. Amari. *Mathematical foundation of neurocomputing.* Proceedings of IEEE, 78:1443-1463, 1990.

[Alander, 1992] J. T. Alander. *On optimal population size of genetic algorithms.* CompEuro' 92. Computer systems and software engineering, Proceedings. Pages: 65-70. 4-8 May 1992.

[Andrews & Patterson, 1976] H. C. Andrews y C. L. Patterson. *Singular value decomposition and digital image processing*. IEEE Trans. Acoust. Speech, Signal Processing, 24:26-53. 1976.

[Bezdek, 1973] J. C. Bezdek. *Fuzzy Mathematics in pattern Classification.* PhD thesis, Applied math. Center, Cornell University, Ithaca, 1973.

[Bezdek, 1981] J. C. Bezdek. *Pattern recognition with fuzzy objective function algorithms.* Plenum press, New York, 1981.

[Broomhead & lowe, 1988] D. S. Broomhead y D. Lowe. *Multivariable functional interpolation and adaptive networks.* Complex systems, 2:321-255, 1988.

[Castillo, 2000] P. A. Castillo. *Optimización de perceptrones multicapa mediante algoritmos evolutivos*. Proyecto de tesis doctoral, Univ. de Granada. España, 2000.

[Chen & Billings, 1992] S. Chen y S. A. Billings. *Neural Networks for nonlinear dynamic system modelling and identification.* Int. Jour. Contr. 56(2):319-346. 1992.

[Chen et al, 1991] S. Chen, C. Cowan, y P. M. Grant. *Orthogonal least squares learning algorithm for training multi-output radial basis function networks.* IEEE Trans. Neural Networks, 2:302-309, 1991.

[Chen et al, 1992] S. Chen, S. A. Billings, y P. M. Grant. *Recursive hybrid algorithm for nonlinear system identification using radial basis function networks.* Int. Jour. Contr. 55(5):1051-1070. 1992.

[Cherkassky & Lari-Najafi, 1991] V. Cherkassky, H. Lari-Najafi. *Constrained topological mapping for non-parametric regression analysis.* Neural Networks, vol. 1,pp 79-84, July 1991.

[Cherkassky et al, 1996] V Cherkassky, D. Gehring, y F. Mulier. *Comparison of adaptive methods for function estimation from samples.* IEEE Trans. Neural Networks, 7(4):969-984, 1996.

[Davis, 1991] L. Davis. editor. *Handbook of genetic algorithms.* Van Nostrand Reinhold, New York, 1991.

[De Jong, 1975] K. A. De Jong. *An analysis of the behaviour of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[De Jong, 1985] K. A. De Jong. *Genetic algorithms: a 10 year perspective.* In (Grefenstette, 1985), pp: 169-177. 1985.

[Dickerson & Kosko, 1996] J. A. Dickerson y B. Kosko. *Fuzzy function approximation with ellipsoidal rules.* IEEE Trans. Syst. Man. and Cyber. Part B, 26(4):542-560. 1996.

[Duda & Hart, 1973] R. O. Duda y P. E. Hart. *Pattern Classification and Scene analysis.* John Wiley, New York, 1973.

[Friedman, 1981] J. H. Friedman. *Projection pursuit regression.* J. American statist. Assoc., 76:817-823, 1981.

[Friedman, 1991] J. H. Friedman. *Multivariate adaptive regression splines (with discussion).* Ann. Statist. 19:1-141, 1991.

[Gersho, 1979] A. Gersho. *Asymptotically Optimal Block Quantization.* IEEE Trans. Inf. Theory, 25(4):373-380. 1979.

[Goldberg, 1989] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley, Reading, MA, 1989.

[Goldberg, 1989b] D. E. Goldberg. *Sizing Populations for serial and parallel genetic algorithms.* In Schaffer 1989, pages 70-79. 1989.

[Goldberg, 1994] D. E. Goldberg. *Genetic and evolutionary algorithms come of age.* Communications of the ACM, 37(3):113-119. 1994.

[Goldfeld et al, 1966] S. M. Goldfeld, R. E. Quandt y H. F. Trotter. *Maximization by quadratic hill climbing.* Econometrica. 34:541-551, 1966.

[Golub & Van Loan, 1996] G. H. Golub y C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, 3rd edition. 1996.

[González et al, 2002] J. González, I. Rojas, H. Pomares, J. Ortega y A. Prieto. *A new clustering technique for function approximation.* IEEE Trans. Neural Networks, volume 13. n.1 January, pp: 132-142, 2002.

[González, 2001] J. González. *Identificación y optimización de redes de funciones de base radial para aproximación funcional.* Tesis doctoral, Dpto. de Arquitectura y Tecnología de Computadores. Universidad de Granada, 2001.

[Hammarling, 1985] S. J. Hammarling. *The singular value descomposicion in multivariate statistics.* ACM Signum Letter, 20:2-25. 1985.

[Haykin, 1994] S. Haykin. *Neural Networks: A comprehensive foundation.* McMillan College Publishing, New York, 1994.

[Hebb, 1949] D. O. Hebb. *The organization of Behaviour.* John Wiley & Sons, New York, 1949.

[Holland, 1975] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press (Second Edition: MIT Press, 1992), 1975.

[Houck et al, 1995] C. R. Houck, J. A Joines, and M.G. Kay, *A genetic algorithm for function optimization: a Matlab implementation*, NCSU-IE TR 95- 9, North Carolina State University, 1995.

[Jang & Sun, 1993] J.-S. Roger Jang y C.-T. Sun. *Functional equivalence between radial basis function networks and fuzzy inference systems. IEEE Transactions on Neural Networks,* 4(1):156-159, January 1993.

[Jang et al, 1997] J-S. Jang, C-T. Sun, E. Mizutani. *Neuro-fuzzy and soft-computing.* Prentice Hall, Upper Saddle River, N.J. 1997.

[Joines & Houk, 1994] J. Joines & C. Houck. *On the use of non stationary penalty to solve constrained optimization problems with genetic algorithms.* IEEE International Symposium Evolutionary Computation, Orlando, Fl, pp. 579-584. 1994.

[Karayiannis & Mi, 1997] N. B. Karayiannis y G. W. Mi. *Growing radial basis function networks: Merging supervised and unsupervised learning with network growth techniques.* IEEE Trans. Neural Networks, 8:1492-1506, November, 1997.

[Kohonen, 1990] T. Kohonen. *The self-organizing map.* IEEE Proceedings, vol. 78, no. 9, pp:1464-1480, September 1990.

[Laub, 1997] A. J. Laub. *Numerical linear algebra aspects of control design applications.* IEEE Trans. Automat. Control, 30:97-108. 1997.

[LeCun et al, 1990] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubard, y L. D. Jackel. *Handwritten digit recognition with a back propagation network.* In D. S. Tourestsky, editor, Neural information processing system, pp: 143-155, Morgan Kauffman, San Mateo, CA, 1990.

[Levenberg, 1944] K. Levenberg. *A method for the solution of certain problems in least squares.* Quart. Apl. Math., 2:164-168, 1944.

[Lippman, 1987] R. P. Lippman. *An Introduction to Computing with Neural Networks.* IEEE Acoustics, Speech, and signal processing magazine, 4(2): 4-22, 1987.

[Lowe, 1989] D. Lowe. *Adaptive radial basis function nonlinearities, and the problem of generalization.* In *Proceedings of the First IEEE International Conference on Artificial Neural Networks,* Pages 171-175, London, UK, 1989.

[Mamdani & Assilian, 1975] E. H. Mamdani & S. Assilian. *An experiment in linguistic synthesis with a fuzzy logic controller.* International Journal of Man-Machine Studies, 7(1):1-13, 1975.

[Marquardt, 1963] D. W. Marquardt. *An algorithm for least squares estimation of neural inequalities.* SIAM J. App. Math., 11:431-441. 1963.

[Michalewicz, 1999] Z. Michalewicz. G*enetic algorithms + data structures = evolution programs.* Springer-Verlag, New York USA, 3 Edition, 1999.

[Moody & Darken, 1989] J. Moody y C. Darken. *Fast learning in networks of locally-tuned processing units.* Neural Computation, 1:281-294, 1989.

[Mouzouris & Mendel, 1996] G. C. Mouzouris y J. M. Mendel. *Designing fuzzy logic systems for uncertain environnment using a singular value QR decomposition method.* In (Council, 1996), pp: 295-301. 1996.

[Musavi et al, 1992] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris y D. M. Hummels. *On the training of the radial basis function classifiers.* Neural Networks, 5(4):595-603, 1992.

[Nie & Lee, 1996] J. H. Nie y T. H. Lee. *Rule based modelling: Fast construction and optimal manipulation.* IEEE Trans. Syst. Man. and Cyber. Part A, 26(6), 1996.

[Park & Sandberg, 1993] J. Park y I. W. Sandberg. *Approximation and Radial Basis Function networks.* Neural Computation, 5:305-316, 1993.

[Poggio & Girosi, 1990] T. Poggio y F. Girosi. *Networks for approximation and learning.* The proceedings of the IEEE, 78(9):1485-1497, September 1990.

[Pomares et al, 2000] H. Pomares, I. Rojas, J. Ortega, J. González, y A. Prieto. *A systematic approach to a self-generating fuzzy rule-table for function approximation.* IEEE Trans. Syst. Manu. and Cyber. Part B, 30(3):431-447, 2000.

[Pomerleau, 1991] D. A. Pomerleau. *Efficient training of artificial neural networks for autonomous navigation.* Neural computation, 3:88-97, 1991.

[Pomerleau, 1992] D. A. Pomerleau. *Neural network perceptron for mobile robot guidance.* PhD thesis department of computer science, Carnegie Mellon University, 1992.

[Rivas, 2003] V. M. Rivas Santos. *Optimización de redes neuronales de funciones base radiales mediante algoritmos evolutivos.* Tesis doctoral, Dpto. de Arquitectura y Tecnología de Computadores. Universidad de Granada, 2003.

[Rivera, 2003] A. J. Rivera Rivas. *Diseño y optimización de redes de funciones de base radial mediante técnicas bio-inspiradas*. Tesis doctoral, Dpto. de Arquitectura y Tecnología de Computadores. Universidad de Granada, 2003.

[Rojas et al, 2000] I. Rojas, H. Pomares, J. González , E. Ros, M. Salmerón, J. Ortega y A. Prieto. *A new radial basis function networks structure: application to time series prediction.* In S. I. Amari, C. L. Giles, M. Gori y V. Piuri editors. Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, volume IV: 449-454, Como, Italy. IEEE Computer Society, 2000.

[Rojas, 2000] I. Rojas, H. Pomares, J. Ortega, y A. Prieto. *Self-organized fuzzy system generation from training examples.* IEEE Trans. Fyzzy Systems, 8(1):23-26. 2000.

[Rosenblatt, 1962] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan, New York, 1962.

[Rovatti & Guerrieri, 1996] R. Rovatti y R. Guerrieri. *Fuzzy sets of rules sor system identification.* IEEE Trans. Fuzzy systems, 4(2):89-102, 1996.

[Rumelhart et al, 1986] D. E. Rumelhart, G. E. Hinton, R. J. Williams. *Learning Internal Representations by Error Propagation.* In D. E. Rumelhart y James L. McClelland, editors, *Parallel distributed processing: explorations in the microstructure of cognition,* volume 1, 8: 318-362. MIT Press, Cambridge, MA., 1986.

[Russo & Patané 1999] M. Russo y G. Patané. *Improving The LBG Algorithm.* Lecture notes on Computer Science, volume 1606, 621-630, Spring Verlag 1999.

[Säckinger et al, 1992] E. Säckinger, B. E. Boser, J. Bromley, Y. LeCun y L. D. Jackel. *Application of the anna neuralnetwork chip to high-speed character recognition.* IEEE Trans. Neural Networks, 3 :498-505, 1992.

[Sudkamp & Hamell, 1994] T. Sudkamp y R. J. Hamell. *Interpolation, Completion and learning fuzzy rules.* IEEE Trans. Syst. Manu. and Cyber, 24(2):332-343, 1994.

[Terrence & Rosenberg, 1986] Terrence J. Sejnowski y Charles R. Rosenberg. *NETtalk: a parallel network that learns to read aloud.* JHU/EECS 86/01, John Hopkins University, 1986.

[Terrence & Rosenberg, 1987] Terrence J. Sejnowski y Charles R. Rosenberg. *Parallel networks that learn to pronounce english text.* Complex systems, 1:145-168, 1987.

[Van Dijk et al, 2000] S. Van Dijk, D. Thierens, y M de Berg. *Scalability and efficiency of genetic algorithms for geometrical applications.* In M. Deb Schoenauer, R. Yao Günter, E. Merelo Lutton & H.P. Schwefel (Eds.), *Lecture Notes in Computer Science 1917: Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature* (pp. 683-692) Berlin Heidelberg: Springer-Verlag. 2000.

[Vandevalle & De Moor, 1988] J. Vandevalle y B. De Moor. *A variety of applications of singular value decomposition and signal processing.* In E. Deprettere, editor,

SVD and signal processing: Algorithms and architectures, Amsterdam. North Holland. 1988.

[Vignaux & Michalewicz, 1991] G. A. Vignaux y Z. Michalewicz. *A genetic algorithm for the linear transportation problem.* IEEE Trans. Syst. Man. and Cyber. 21(2):445-452. 1991.

[Wang & Mendel, 1992] L. X. Wang y J. M. Mendel. *Generating fuzzy rules by learning from examples.* IEEEb Trans Syst. Man Cybernetics, 22(6), pp: 1414-1427, 1992.

[Wang & Mendel, 1992b] L. X. Wang y J. M. Mendel. *Fuzzy basis function, universal approximation, and orthogonal least squares learning.* IEEE Trans Neural Networks, 3:807-814, 1992.

[Whitehead, 1996] B. Whitehead, T. Choate. *Cooperative competitive genetic evolution of radial basis function centres and widths for time series prediction.* IEEE Trans. On Neural Networks, Vol 7, Nº 4, pages: 869-880. July 1996.

[Widrow & Hoff, 1960] B. Widrow, J. R. Hoff. *Adaptive switching circuits.* In IRE WESCON Convention Record, 96-104, New York, 1960.

[Yen & Wang, 1996] J. Yen y L. Wang. *An SVD-based fuzzy model reduction strategy.* In (Council, 1996), pp: 835-841. 1996.

[Yen & Wang, 1999] J. Yen y L. Wang. *Simplifying rule-based models using orthogonal transformation methods.* IEEE Trans. Syst. Manu. and Cyber. Part B, 29(1):13-24. 1999.

[Zadeh, 1965] L. A. Zadeh. *Fuzzy sets.* Information and d control, 8:338-353, 1965.

[Zadeh, 1971] L. A. Zadeh. *Quantitative fuzzy semantics.* Information sciences, 3:159-176, 1971.

[Zadeh, 1971-bis] L. A. Zadeh. *Similarity relations and fuzzy ordering.* Information sciences, 3:177-206. 1971.

[Zadeh, 1973] L. A. Zadeh. *Outline of a new approach to the analysis of complex systems and decision processes.* IEEE Transactions on systems, Man, and Cybernetics, 3(1): 28-44, January 1973.

[Zadeh, 1975] L. A. Zadeh. *The concept of a linguistic variable and its application to approximate reasoning,* Parts1, 2 and 3. *Information sciences,* 8:199-249, 8:301-357, 9:43-80, 1975.