UNIVERSIDAD DE GRANADA

Departamento de Estadística e Investigación Operativa

Métodos de reducción de Redes Neuronales RBF usando la descomposición QLP para aplicaciones de ajuste de datos y clasificación

TESIS DOCTORAL

Doctorando: Edwirde Luiz Silva

Directores: Andrés González Carmona y Paulo J. G. Lisboa

Granada, junio de 2007

Editor: Editorial de la Universidad de Granada
Autor: Edwirde Luiz Silva
D.L.: Gr. 1445 - 2007
ISBN: 978-84-338-4392-0

Métodos de reducción de Redes Neuronales RBF usando la descomposición QLP para aplicaciones de ajuste de datos y clasificación

Memoria que para optar al grado de Doctor en Estadística e Investigación Operativa presenta el licenciado Edwirde Luiz Silva

Vº Bº de los directores

Andrés González Carmona

Paulo J. G. Lisboa

Departamento de Estadística e Investigación Operativa
UNIVERSIDAD DE GRANADA
2007

Agradecimientos

En primer lugar doy gracias al Dios desconocido y a mi familia.

En segundo lugar, deseo agradecer la ayuda recibida por parte de diversas personas de la Universidad de Granada y de la John Moore University (Liverpool - UK); que de forma directa o indirecta han contribuido a la elaboración de esta tesis, en especial a mis directores de tesis, Dr. Andrés González Carmona, y Dr. Paulo J.G. Lisboa. Quiero destacar de forma especial la ayuda recibida de todos mis compañeros de trabajo de la Universidad de Granada y de la John Moore University, tanto a nivel científico como personal.

Edwirde Luiz Silva

Capítulo 1

Introducción a las Redes Neuronales

1.1. Introducción

Las redes neuronales artificiales (RNA) son algoritmos implementados en forma de programa informático o modelo electrónico, basados en el funcionamiento del cerebro humano. Es un hecho bien conocido que los hombres pueden solucionar fácilmente muchos problemas, difíciles de resolver para las computadoras. El cerebro es un computador (sistema de procesamiento de información) altamente complejo, no lineal y con capacidad paralela de organizar sus constituyentes estructurales, conocidos como neuronas, de forma que realice algunos procesamientos (p.e., reconocimiento de patrones, percepción y control motor) mucho más rápidamente que un computador actual. Una neurona en desarrollo es sinónimo de un cerebro mutable: ese cambio permite que el sistema nervioso en desarrollo se adapte al ambiente. Así como la elasticidad parece ser esencial para el funcionamiento de las neuronas como unidades de procesamiento de información del cerebro humano, también lo es para las redes neuronales constituidas con neuronas artificiales. En la práctica, de forma más general, una red neuronal es una máquina diseñada para modelar la manera en que el cerebro realiza una tarea particular o función de interés; la red se implementa normalmente utilizando componentes electrónicos y/o simulados. Una red neuronal se parece al cerebro en dos aspectos:

1. El conocimiento es adquirido por la red a partir de su ambiente a través de un proceso de aprendizaje.

2. Las fuerzas de conexión entre neuronas, conocidas como pesos sinápti-

cos, son utilizadas para almacenar el conocimiento adquirido.

El procedimiento utilizado para realizar el proceso de aprendizaje es llamado algoritmo de aprendizaje, y su función es modificar los pesos sinápticos de la red de forma ordenada para alcanzar un objetivo definido.

La modificación de los pesos sinápticos es un método tradicional para el diseño de redes neuronales. Este procedimiento es bastante similar a la teoría de los filtros adaptativos lineales, que está bien establecida y que fue aplicada con éxito en diversas áreas, [99, 35]. Sin embargo, es posible también para una red neuronal modificar su propia topología, lo que viene motivado por el hecho de que las neuronas del cerebro humano pueden morir y nuevas conexiones sinápticas pueden crecer.

Con el crecimiento de este área, los modelos neuronales –ahora con múltiples capas– fueron aplicados con éxito a una gran cantidad de problemas. Ese éxito atraía a muchos investigadores de las más diversas áreas, que se interesaron en el estudio de las redes neuronales artificiales. Entretanto, esa explotación del interés por las redes neuronales no se justifica únicamente por el hecho de que se trate de poderosas herramientas de computación. Sin duda, gran parte de esa atención fue despertada por la oferta de una metáfora computacional para el funcionamiento del cerebro y –presumiblemente– de la mente humana [35].

Esa afirmación parece muy clara cuando se observa que muchos de los conceptos "inventados" por la comunidad de computación neuronal ya venían siendo utilizados –aunque no con el mismo significado biológico– en estudios estadísticos, sin que por ello llamasen la misma atención [82]. Esa constatación abrió una doble vía: así como la Estadística tenía mucho que ofrecer a los estudiosos de las redes neuronales, los avances teóricos alcanzados en los estudios de las redes neuronales artificiales generaban frutos originales que también podrían ser absorbidos por los estadísticos. De esa forma, se hizo interesante establecer un canal de comunicación entre las dos áreas: *red neuronal* es lo mismo que *modelo*, *conjunto de entrenamiento* equivale a las *observaciones*; *aprender* es lo mismo que *estimar* parámetros –que también pueden ser llamados *pesos sinápticos* [68]–. En especial, hibridar los métodos estadísticos estructurados con una red neuronal parece ser una estrategia interesante. Sobre todo al incorporar a esos modelos cuestiones interdisciplinares (como su plausibilidad biológica, por ejemplo); esa postura ilustra una representación gráfica y una terminología atrayente para un área repleta de términos antes desconocidos para investigadores provenientes de otras disciplinas.

Las redes de neuronas de base radial han sido aplicadas a una gran variedad de problemas, aunque es necesario señalar que su aplicación no ha sido

tan extendida como en el caso del Perceptrón multicapa. Sin embargo, se han utilizado en diferentes campos, como análisis de series temporales [13, 45], procesamiento de imagen [52], reconocimiento automático del habla [65], etc.

Sin embargo, en cuanto a la primera fase (de posicionamiento de las neuronas individuales en la capa oculta y reducción de la matriz de diseño (centros y radios), o determinación de la propia cantidad de neuronas a emplear) no se han presentado algoritmos suficientemente potentes, en el sentido de que tradicionalmente se ha venido recurriendo a un simple *clustering* sobre los datos de entrada (llegando incluso al caso extremo de colocar una neurona individual en cada dato de entrada) o a estrategias heurísticas[1] de dudosa fundamentación teórica.

Los algoritmos que se proponen en este trabajo optimizan procedimientos para la reducción del número de neuronas existente en la capa oculta, realizando dicha reducción con atención especial a la significación de cada una de ellas, y preservando las que se consideran como más relevantes o importantes, según algunos criterios estadísticos que se fundamentarán de forma teórica en los capítulos sucesivos, atendiendo también a la complejidad computacional de los correspondientes algoritmos, que puede resultar crítica a la hora de idear aplicaciones que operen con conjuntos de datos disponibles en tiempo real.

Aún más, los mismos procedimientos que permitirán reducir el tamaño de la capa oculta (es decir, el número de neuronas en ella) nos serán también útiles para reducir la dimensión del espacio de entradas a la red, es decir, procurarán determinar cuáles de las entradas son las importantes según la descomposición QLP a la hora de ajustar el modelo neuronal, luchando de este modo contra la conocida *maldición de la dimensionalidad*, puesto que menos entradas significa un espacio de entradas exponencialmente menor y, por ende, se necesita un número menor de neuronas para llenar el espacio de datos de entrada (o su equivalente tras el mapeo no lineal) a la hora de predecir o ajustar dichos datos. Por tanto, la ventaja de los métodos reductores que introduciremos es doble: permite redes con menos entradas, y también menos neuronas y, en general, redes más parsimoniosas, lo cual suele conducir (siempre y cuando el aprendizaje se lleve a cabo de forma correcta) a obtener el mismo rendimiento predictivo (y a veces incluso mejor) pero con menores recursos de cómputo y de forma más eficiente.

[1]El método heurístico es una tecnología de programación que dentro de sus rutinas de detección y eliminación de especies virales, incluye cadenas típicas que son similares, parecidas o afines a virus auténticos. El método heurístico no está bien programado, es susceptible de incurrir en resultados de falso positivo o falso negativo. Es un algoritmo que utiliza pruebas, exámenes o aproximaciones para llegar a dar con una solución

Como se mencionó en la introducción de este texto, en los capítulos que siguen veremos ejemplos relativos a campos tan diversos como: la predicción de series caóticas, el contraste de hipótesis de una función de densidad con parámetros particulares que hacen difícil diferenciarla, el ajuste de funciones no lineales, la determinación eficiente del rango numérico de matrices de datos contaminadas por ruido, e incluso estudiaremos la predicción de datos procedentes de un sistema caótico determinista, problema este ante el que las técnicas más tradicionales de ajuste de modelos han resultado claramente insuficientes. Usaremos la reducción de neuronas RBF para alcanzar estos objetivos específicos.

Somos conscientes, a pesar de todo, de que escasean en el campo de las redes neuronales, los estudios teóricos sobre su ajuste y rendimiento. Al tratarse de modelos sin los presupuestos estadísticos habituales, la derivación y el análisis de rendimiento de los algoritmos neuronales carece, por ejemplo, de estimaciones a priori del error de aproximación, o de conclusiones que requieran un enfoque asintótico o relativo a la distribución particular de los datos de salida o del posible error neuronal.

Descomposición QLP para reducción de neuronas RBF

Nuestro principal interés reside en el uso de redes neuronales reducidas por QLP para predicción y clasificación de diferentes densidades de probabilidades. Una de nuestras propuestas ha sido basar las mejoras algorítmicas en el uso de descomposiciones matriciales de tipo ortogonal (como la descomposición QR y QLP) y se demostrará de forma teórica la bondad de esta descomposición para determinar modelos neuronales con mejor comportamiento y más reducidos (4).

Un marco general para la clasificación

Considérese la figura 1.1, en ella pueden observarse los principios de operación del modelo general que proponemos en esta memoria para abordar la reducción de neuronas RBF por QLP. También se muestra de forma gráfica el procedimiento de cómo se han reducido neuronas RBF dentro del algoritmo **newrb** del entorno Matlab® . R es el número de elementos de entrada, $S1$ es el número de neuronas en la primera capa oculta, y $S2$ es el número de neuronas en la segunda capa, $b1$ y $b2$ son los sesgos de la primera y segunda capa respectivamente. Se puede verificar que la reducción (*pruning*) de la matriz de diseño (compuesta de entradas y centros) en la primera capa influye en la segunda capa en los coeficientes $LW^{2,1}$. Las lagunas provocadas por la reducción son rellenadas con ceros en los coeficientes $LW^{2,1}$, y pos-

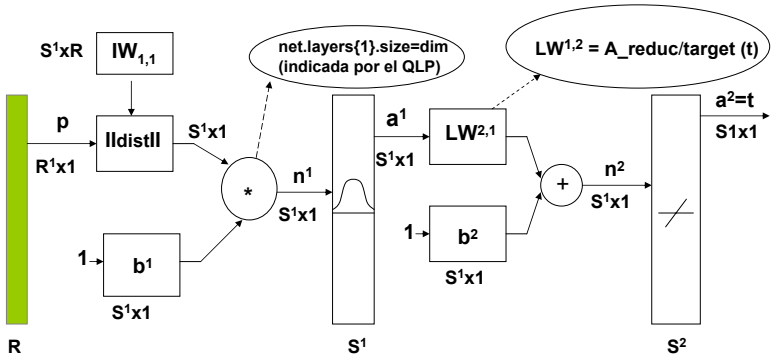


Figura 1.1: Relación entre los distintos parámetros de **newrb** en Matlab®

teriormente estos ceros son descartados para evitar cómputos innecesarios. La ecuación de los coeficientes de la segunda capa es $W_{2,1} = \frac{1}{t} A_{\text{reduc}}$ donde A_{reduc} provoca la reducción de la matriz de diseño en la capa oculta.

Parámetros de la RBF (newrb)

Los parámetros generales que controlan el funcionamiento del algoritmo RBF reducido por QLP, dentro del entorno Matlab® , usados en el experimento de predicción de serie caótica y clasificación de densidades de probabilidad se detallan en la tabla siguiente:

Notación	Significado
R	Vector de entradas a la red
$IW_{1,1}$	Centro y peso de la primera capa
b^1	Sesgo de la primera capa
$*$	Producto red
$a1$	Salida de la primera capa
$LW^{2,1}$	Coeficientes de la segunda capa
$A_{\text{reduc}}/\text{target}(t)$	Reducción de los coeficientes $LW^{2,1}$
S^2	Función de transferencia lineal
$a^2 = t$	Salida de la red de RBFs

En la figura 1.1 se puede observar que en la segunda capa oculta $LW^{2,1}$ se ha reducido el número de neuronas a través de la descomposición QLP de tal forma que con menos neuronas se puede tener una respuesta satisfactoria en problemas de clasificación (4.6) y también de predicción de la serie caótica de Henón (4.5).

Un marco general para la descomposición QLP para la función SinE

En los problemas de ajustes de una función SinE se ha creado una RBF desde el principio a través del algoritmo rbfQLP, ver tabla 1.1, en la que se ha hecho un *pruning* en la matriz de diseño de diversos tipos en cuanto a la naturaleza de la RBF, estos tipos pueden ser: gaussiano, multicuádratico, Cauchy e inversa-multicuadrática. En la figura 1.2 se ha mostrado el *pruning* de la matriz H para el problema de ajuste de la función especifica **SinE** (4.4)

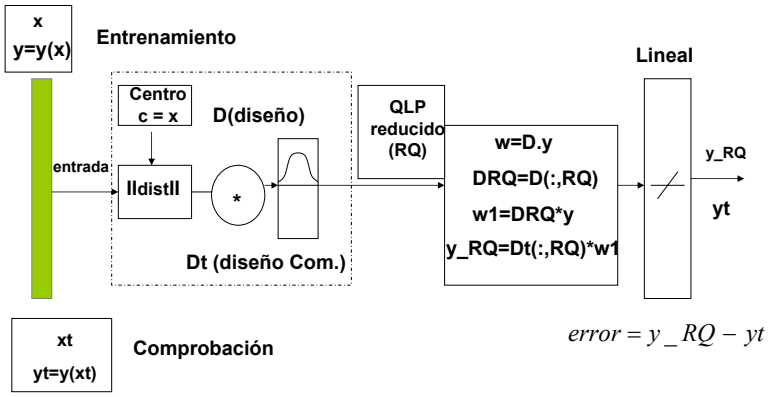


Figura 1.2: Modelo de reducción neuronal RBF para el ajuste de **SinE**

En la figura 1.2 se pueden observar los principios de operación del modelo general rbfQLP que proponemos en este trabajo para abordar la predicción de la función específica, **SinE**. El modelo considera entradas x e y para entrenamiento, y xt e yt para comprobación.

El núcleo fundamental lo constituye un modelo neuronal basado en funciones radiales Gaussiana, Cauchy, Multicuadrática o Multicuadrática inversa. La selección de las neuronas se lleva a cabo usando la descomposición QLP. Por ultimo, las predicciones efectuadas fueron hechas con neuronas con el objetivo de encontrar un modelo mejor para dichas predicciones (4.4).

Estructura de la memoria

La distribución de los temas a abordar en este trabajo es la siguiente:

Notación	Significado
(x, y)	Vector de entrada para entrenamiento
(xt, yt)	Vector de entrada para comprobación
D	Matriz de diseño
Dt	Matriz de diseño para comprobación
RQ	Orden del QLP para reducción de D y Dt
$w1$	Peso ajustado según la reducción QLP
DRQ	Reducción de la matriz de diseño reducida
y_{RQ}	Aproximación de la red reducida
yt	Objetivo
S^2	Función de transferencia lineal
$a^2 = t$	Salida lineal de la red de RBF

Cuadro 1.1: Secuencia del algoritmo rbfQLP

- En este capítulo, 1.2, se proporciona una relación entre las redes neuronales y la Estadística. También en este capítulo se muestra la naturaleza estadística del proceso de aprendizaje y el paralelismo entre la Estadística y las redes neuronales artificiales. Es por tanto natural esperar que los métodos que emplean redes neuronales artificiales sean de algún modo paralelos (o similares) a los clásicamente empleados en la Estadística moderna. También en este capítulo se tratarán las derivaciones matemáticas que conciernen al uso de la arquitectura de funciones de base radial y que constituyen el principal componente de los esquemas predictivos y clasificativos que se propondrán. Los cálculos matemáticos básicos considerados necesarios se introducen detalladamente, teniendo en cuenta el contexto del problema a resolver en este trabajo.

- El capítulo 2 se ocupa de la funciones de base radial. Se mostrará cómo funcionan las neuronas RBF y sus operaciones divididas en partes, así como las nociones matemáticas de las RBF y las funciones de activación típicas de una RBF, tales como la gaussiana y las multicuadráticas. También en esta sección trataremos la teoría de la regularización y una aplicación a una regresión estadística así como una interpretación de las RBF en tres dimensiones. Posteriormente, en 2.3, se ocupa de la descomposición QLP en comparación con otras descomposiciones tradicionales tales como SVD y QR. También en este capítulo trataremos los cálculos matriciales necesarios que conciernen al uso del QLP. El QLP se mostrará como una alternativa al SVD, pues este presenta un

tiempo más largo en comparación con otras descomposiciones. Se utilizará en la matriz de retardos (*lags*) para predicciones de series caóticas. También se abordan estrategias de paralelización de las rutinas SVD, QR y QLP, con el objetivo de reducir el tiempo de cómputo necesario para realizar dichas descomposiciones. Los resultados de ejecución de ambas implementaciones, es decir, las diversas descomposiciones, han mostrado que el QLP suministra una ganancia de velocidad y eficiencia significativas en los algoritmos recursivos [16]. Finaliza el capítulo en 2.4, donde se estudia la aplicación a tres densidades de probabilidad —la Gamma, la Weibull y la Lognormal— que ofrecen cierta dificultad de identificación para la misma media y coeficiente de variación.

- El capítulo 3 contiene los algoritmos desarrollados en esta memoria.
- El capítulo 4 presenta con detalle resultados experimentales, añadiendo nuevos ejemplos y resultados a aquellos usados como ilustración en los capítulos previos así como las conclusiones de esta tesis.
- El capítulo 5 presenta en inglés (para cumplir el requisito exigido para las tesis de doctorado europeo) dos apartados: Differentiating features for the F distributions with different degrees of freedom through RBF network pruning with QLP and Using RBF reduced by QLP decomposition for Probability Density Estimation.

Finalmente, se incluyen un apéndice dedicados a exponer los conceptos y fundamentos teóricos que sustentan el desarrollo de los algoritmos de RBF reducido por QLP y métodos que se describirán. Fundamentalmente, por una parte, los conceptos fundamentales de algunos resultados del álgebra matricial, así como un exposición de los principios de operación de la descomposición QLP, así como el tratamiento de esta descomposición en el entorno Matlab® . Por otra parte, se dedica un apartado a las redes de neuronas RBF y cómo la función de coste, 2.19, puede ser reescrita en forma matricial. Se ocupa de los cálculos de los pesos y umbrales de las neuronas de salida de la red en una fase supervisada. Se calculan también los pesos, umbrales, centros y amplitudes del método de aprendizaje totalmente supervisado.

1.1.1. Objetivo principal

1. Contribuir al diseño de redes artificiales óptimas de tipo RBF.
2. Reducir las neuronas RBF (dentro del entorno Matlab®) y también reducir la matriz de diseño (compuesta de entradas y centros) a través

de la descomposición QLP, con el objetivo de hacer ajustes y clasificaciones con una RBF reducida.

Objetivos específicos

1. Usar la técnica de descomposición QLP para identificar las entradas y neuronas RBF principales
2. Reducir las neuronas en la capa oculta de la RBF.
3. Comparación de la reducción frente a la no reducción de las neuronas.
4. Verificar la veracidad del ajuste, predicción o contrastes efectuados con RBF reducida a través de herramientas estadísticas.

1.1.2. Paradigmas de aprendizaje de RNA

En la actualidad se están realizando numerosas investigaciones y propuestas relacionadas con muchos tipos de redes neuronales artificiales, y cada año los investigadores especializados crean nuevas arquitecturas, paradigmas y algoritmos de aprendizaje, o mejoras de los ya existentes. Veamos el tipo de aprendizaje utilizado en este trabajo.

Aprendizaje supervisado

En el aprendizaje supervisado un *maestro* guía la red en cada etapa del aprendizaje, indicándole el resultado correcto [9]. La misión del algoritmo es ajustar los pesos de la red de manera tal que, dado un conjunto de entradas, las salidas proporcionadas por la red deberán coincidir lo más posible con las salidas especificadas en el patrón de entrenamiento. En esta investigación se utilizarán tambien algoritmos de entrenamiento supervisados, debido a su adecuación para resolver problemas de predicción y clasificación.

En función de la cantidad de error detectado se lleva a cabo la modificación de los parámetros libres de las neuronas y de la red. Esto se hace mediante la aplicación de reglas o ecuaciones recursivas que se derivan considerando elementos tales como el gradiente de la función de error con respecto a dichos parámetros. Tras un cierto período repitiendo este procedimiento con diversos patrones presentados en orden aleatorio, se espera que la respuesta actual de la red pueda converger en algún sentido especificado a la clase de respuesta que se está esperando que produzca. Es importante por tanto que el proceso no degenere en un comportamiento inestable, para lo cual se deben elegir con sumo cuidado las constantes que pudiesen existir asociadas al propio algoritmo de ajuste secuencial.

En esta fase se calculan los pesos y umbrales de las neuronas de salida de la red. En este caso, el objetivo es minimizar las diferencias entre las salidas de la red y las salidas deseadas. Por tanto, el proceso de aprendizaje está guiado por la minimización de una función error computada en la salida de la red, 2.26 y 2.16.

Fase supervisada: Matriz pseudoinversa Debido a que la salida de la red, 2.11, depende linealmente de los pesos y umbrales, otro método para el cálculo de dichos parámetros es el llamado método de la pseudoinversa [4]. Se trata de un método que proporciona una solución directa al problema de optimización. Dicha solución viene dada por la siguiente expresión matricial:

$$\theta \cdot \mathbf{w} = \mathbf{d} \ , \tag{1.1}$$

La inversa sería:

$$\mathbf{w} = \theta^{-1} \cdot \mathbf{d} \ , \tag{1.2}$$

donde W es la matriz de pesos y umbrales de la RBF, de orden $(m+1) \times r$, de modo que:

$$W = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1r} \\ W_{21} & W_{22} & \cdots & W_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ W_{m1} & W_{m2} & \cdots & W_{mr} \\ u_1 & u_2 & \cdots & u_r \end{pmatrix} \tag{1.3}$$

θ es una matriz de orden $N \times (m+1)$ que contiene las activaciones de las neuronas ocultas de la red para los patrones de entrada:

$$\theta = \begin{pmatrix} \theta_1(1) & \theta_2(1) & \cdots & \theta_m(1) & 1 \\ \theta_1(2) & \theta_2(2) & \cdots & \theta_m(2) & 1 \\ \vdots & \vdots & \ddots & \vdots & \\ \theta_1(N) & \theta_2(N) & \cdots & \theta_m(N) & 1 \end{pmatrix} \tag{1.4}$$

donde $\theta_i(n)$ es la activación de la neurona oculta, i, para el patrón de entrada, $X(n)$; y d es la matriz de salidas deseadas para la red, de orden $N \times r$,:

$$d = \begin{pmatrix} d_1(1) & d_2(1) & \cdots & d_r(1) \\ d_1(2) & d_2(2) & \cdots & d_r(2) \\ \vdots & \vdots & \ddots & \vdots \\ d_1(N) & d_2(N) & \cdots & d_m(N) \end{pmatrix} \tag{1.5}$$

donde $d_k(n)$ es la coordenada, k, de la salida deseada para el patrón, $X(n)$.

En el experimento **SinE** los pesos y umbrales de la red se obtienen de esta forma; o sea, calculando la pseudoinversa de la matriz θ, con lo que se obtiene una solución óptima al problema de minimización. Sin embargo, es necesario señalar que, aunque dicho método proporciona una solución directa al problema y no iterativo, desde un punto de vista práctico no es precisamente el método más eficiente, pues el cálculo de la matriz pseudoinversa debe realizarse mediante métodos numéricos [29, 35], los cuales podrían requerir un alto coste computacional y ocasionar errores debido a problemas de precisión [44]. Por tanto, en el contexto de las RBF, el método más utilizado para la determinación de pesos y umbrales es el algoritmo de los mínimos cuadrados.

Aprendizaje híbrido

El método híbrido realiza el aprendizaje de las redes de base radial en dos fases. La primera es la fase no supervisada, que consiste en la determinación de los centros y amplitudes de las neuronas de la capa oculta, y la segunda es la fase supervisada, que consiste en la determinación de pesos y umbrales de la capa de salida [44]. En los experimentos SinE y RBF reducida, recogidos en el capítulo 4, para clasificar tres densidades de probabilidad, se usará el método de aprendizaje híbrido, pues los centros y las anchuras serán determinados por el método no supervisado y la determinación de los pesos y umbrales por el método supervisado.

Fase no supervisada Puesto que las neuronas ocultas de las redes de base radial se caracterizan porque representan zonas diferentes del espacio de patrones de entrada, los centros y las desviaciones de las funciones de base radial deben ser determinados con este objetivo, es decir, con el objetivo de clasificar el espacio de entrada en diferentes clases. El representante de cada clase será el centro de la función de base radial y la desviación vendrá dada por la amplitud de cada clase.

Para los parámetros de la capa oculta –centros y desviaciones– el proceso de aprendizaje debe estar guiado por una optimización en el espacio de patrones de entrada, pues cada una de las neuronas ocultas en la red de base radial va a representar una zona diferente del espacio de entrada. Aquí, los centros serán escogidos de modo fijo, de forma que estén en la vecindad del patrón.

Una vez determinados los centros de las funciones de base radial, para el cálculo de las desviaciones serán consideradas varias amplitudes, de manera que cada neurona oculta se active en una región del espacio de entrada y de

manera que el solapamiento de las zonas de activación de una neurona con otra sea lo más ligero posible, para suavizar así la interpolación.

1.2. Relación entre Redes Neuronales y Estadística

La capacidad de aprender a través de ejemplos y de generalizar la información aprendida es el atractivo principal de la solución de problemas a través de RNA [35]. La generalización (que está asociada a la capacidad de la red de aprender de un conjunto más o menos extenso de ejemplos y, posteriormente, producir respuestas coherentes para datos no conocidos) es una demostración de que la capacidad de las RNA va mas allá de la simple captación de relaciones de entrada-salida. Las RNA son capaces de extraer informaciones no presentadas de forma explícita, a través de ejemplos previos [35].

La utilización de una RNA en la solución de una tarea pasa, ante todo, por una fase de aprendizaje, cuando la red se autoorganiza (o automodifica) para extraer informaciones relevantes contenidas en patrones de información que le son presentados, creándose así una representación interna propia para el problema.

Esta etapa de aprendizaje consiste técnicamente en un proceso interactivo de ajuste secuencial o recursivo de parámetros de la red (los pesos de las conexiones entre las unidades de procesamiento) que de esta forma se considera que guardan, al final del proceso, el conocimiento que la red adquirió del ambiente en el que estuvo operando. Dicho conocimiento está almacenado de forma implícita, esto es, normalmente el valor exacto de los pesos individuales no nos dice nada concreto (no contiene información exactamente localizada del problema) sino que es más bien la operación global de la red, entendida como un bloque con entradas y salidas, la que produce el efecto deseado de aproximación, predicción, o ajuste de los datos que esta ha contemplado. De esta forma, la información se encuentra paralelamente distribuida, como anteriormente se indicó.

Finalmente, como salida o salidas globales de la red, encontraremos neuronas o elementos que suelen efectuar un agregamiento (o combinación generalmente lineal) de las respuestas neuronales (es decir, las salidas de los elementos individuales en la capa inmediatamente anterior). Generalmente sólo se dispone de una salida, que puede representar una respuesta en un rango contínuo y que puede normalizarse, truncarse, distribuirse en varias categorías, etc, dependiendo (como en el caso de la elección de las entradas

globales a la red) de la formulación y características concretas del problema que se esté intentando resolver.

1.2.1. Naturaleza estadística del proceso de aprendizaje

Esta sección trata de los aspectos estadísticos del aprendizaje. En esta discusión no nos interesa la evolución del vector de pesos, $\mathbf{w}$, mientras la red neuronal se somete a un algoritmo de aprendizaje. Ahora, nos concentramos en la desviación o discrepancia entre una función, $f(\mathbf{X})$, y la función real, $F(\mathbf{x}, \mathbf{w})$, realizada por la red neuronal, donde el vector $\mathbf{X}$ representa el conjunto de señales de entrada. La desviación es expresada en términos estadísticos. En la red neuronal el conocimiento empírico sobre un fenómeno físico o ambiente de interés puede ser codificado a través de entrenamiento. Por conocimiento empírico entendemos un conjunto de medidas que caracterizan el fenómeno.

Formulación estocástica del problema del aprendizaje

Para ser más específicos, se considera un fenómeno estocástico general, descrito por un vector aleatorio, $\mathbf{X}$, el cual consiste en un conjunto de variables en principio independientes, y un escalar aleatorio, D, que representa una variable dependiente de las anteriores. Los elementos del vector aleatorio, $\mathbf{X}$, pueden tener significados físicos particulares diferentes. La suposición de que la variable dependiente, D, es escalar se hace simplemente para simplificar la exposición, sin pérdida de generalidad.

Supongamos también que se tiene una muestra de P realizaciones del vector aleatorio, $\mathbf{X}$, denotadas[2] por $\{\mathbf{X}_i\}_{i=1}^{P}$, y un conjunto correspondiente de realizaciones del escalar aleatorio, D, representadas por $\{d_i\}_{i=1}^{P}$. Estas realizaciones (o medidas) constituyen nuestro conjunto de entrenamiento, Ψ:

$$\Psi \equiv \{(\mathbf{X}_i, d_i)\}_{i=1}^{P} \ . \tag{1.6}$$

Normalmente, no conocemos la relación funcional exacta entre $\mathbf{X}$ y D, por lo cual proponemos el modelo general siguiente [96]:

$$D = f(\mathbf{X}) + \epsilon \ , \tag{1.7}$$

donde $f(\cdot)$ es una función determinística de su argumento vectorial, y ϵ representa un término de error aleatorio, que representa nuestra falta de conocimiento sobre la dependencia entre D y $\mathbf{X}$. El modelo estadístico descrito

[2]En esta discusión seguimos la notación empleada en [35].

en la ecuación (1.7) se conoce como *modelo regresivo*, y se muestra de forma esquemática en la figura 1.3 [34].

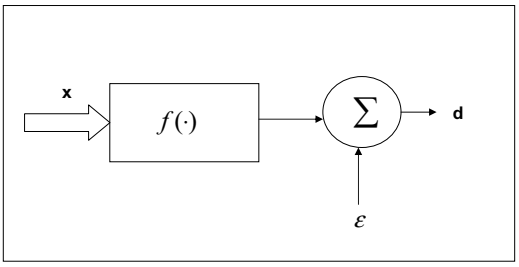


Figura 1.3: Modelo matemático regresivo.

El término de error, ϵ, es, en general, una variable aleatoria con media nula y con probabilidad de ocurrencia positiva (soporte no vacío). Más concretamente, este modelo regresivo presupone las siguientes propiedades [35]:

1. El valor medio del término de error, dada cualquier realización concreta de $\mathbf{x}$, es cero; esto es:

$$E[\epsilon|\mathbf{X} = \mathbf{x}_i] = 0, \text{ para todo } i \ , \tag{1.8}$$

   donde E es el operador estadístico del valor esperado. Como un corolario de esta propiedad, podemos afirmar que la función de regresión, $f(\mathbf{x})$, es la media de la salida del modelo, D,

$$f(\mathbf{x}) = E[D|\mathbf{x}] \ . \tag{1.9}$$

2. El término de error, ϵ, no está correlacionado con la propia función de regresión, o sea:

$$E[\epsilon f(\mathbf{X})] = 0 \ . \tag{1.10}$$

   Esta propiedad es conocida como *principio de ortogonalidad*, que afirma que toda la información sobre D que se puede extraer a partir de las entradas, $\mathbf{X}$, se encuentra codificada en la función de regresión, f [35].

   Es sencillo demostrar esta propiedad sin más que tener en cuenta que

$$E[\epsilon f(X)] = E[E[\epsilon f(X)|x]] = E[f(X)E[\epsilon|x]] = E[f(X) \cdot 0] = 0$$

El modelo regresivo, figura 1.3, es una descripción matemática de un ambiente estocástico. Por otro lado, la figura 1.4, corresponde a un modelo que se basa en una red neuronal codificada con el conocimiento empírico.

Así, la red neuronal suministra una aproximación para el citado modelo regresivo. Supongamos que la respuesta real de la red, producida en respuesta al vector X, está representada por la variable aleatoria $Y, Y = F(X, w)$, donde $F(\cdot, w)$ es la función de entrada–salida realizada por la red neuronal. Conocidos los datos de entrenamiento, Ψ, el vector de pesos, w, se obtiene mediante minimización de la función de costo,

$$\xi(w) = \frac{1}{2}\sum(d_i - F(x_i, w))^2 \tag{1.11}$$

donde el factor $\frac{1}{2}$ se utiliza por consistencia con notaciones posteriores. Con excepción de dicho factor, la función de coste, $\xi(w)$, es la diferencia cuadrática entre la respuesta deseada, d, y la respuesta real, y, de la red neuronal, calculada como la media sobre todo el conjunto de datos de entrenamiento, Ψ.

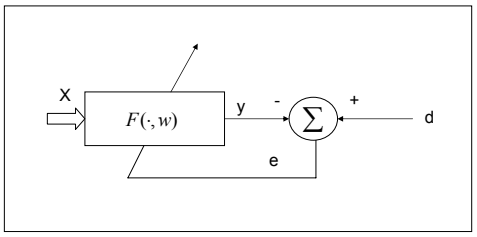


Figura 1.4: Modelo físico de la red neuronal

Paralelismo entre los modelos estadísticos y los neuronales

Se sabe que la Estadística comprende un conjunto de métodos que sirven para recoger, organizar, resumir y analizar datos, así como para extraer conclusiones y tomar decisiones. Es por tanto natural esperar que los métodos que emplean redes neuronales artificiales sean de algún modo paralelos (o similares) a los clásicamente empleados en la Estadística moderna.

El uso de las redes neuronales creció fuertemente a mediados de los años 80, y existen diversas modelos equivalentes a otros estadísticos, [82], que recogemos en el cuadro 1.2.1:

Modelo Neuronal	**Estadística**
Aprendizaje	Estimación
Pesos	Parámetros
Generalización	Interpolación
Conjunto de entrenamiento	Observación
Entradas	Variables independientes
Salidas	Variables dependientes
Perceptrón simple (nodo de tipo umbral)	Análisis discriminante
Perceptrón simple (nodo sigmoidal)	Regresión logística
Adalina	Regresión lineal
Perceptrón multicapa	Regresión no lineal simple y Regresión no lineal multivariada
Aprendizaje hebbiano no supervisado	PCA
Red simple de Kohonen (competitiva)	Análisis cluster Mínimos cuadrados
Cuantificación de vectores LVQ	Análisis discriminante (vecindad)
Funciones de base radial (RBF)	Regresión basada en núcleos (*kernels*)

Cuadro 1.2: Relaciones entre modelos y técnicas neuronales y estadísticas.

En [82] se comenta que los expertos en Estadística critican las redes neuronales porque sus algoritmos de entrenamiento son poco eficientes, muy lentos, no tienen aún un fundamento teórico suficientemente sólido, y necesitan mucho ajuste heurístico de parámetros (*ratio* de aprendizaje, número de neuronas ocultas, etc). Incluso se citan métodos estadísticos de aprendizaje

aplicados al perceptrón multicapa que resultan mucho más rápidos que el clásico *backpropagation*.

Pensamos que este punto de vista es un tanto extremista. Ni las redes neuronales son tan excelentes como en algún momento se ha tratado de mostrar, ni poseen tantos aspectos negativos como algunos estadísticos sugieren. En [24] se observa un análisis curioso de la amistad entre expertos neuronales y estadísticos. Ambas técnicas tratan de resolver en ocasiones problemas similares, de modo que resulta lógico que se llegue a soluciones semejantes; por este motivo, parece más adecuada la cooperación entre ambas que el enfrentamiento. En este sentido, las redes neuronales pueden beneficiarse de la Estadística en numerosos aspectos, como por ejemplo, el empleo de técnicas estadísticas para el análisis de la relevancia de las variables de entrada, su utilización en una inicialización más inteligente de los pesos o en el análisis de la operación de la red. Un estudio comparativo de ambas técnicas muy interesante aparece en [96] y en [95].

Pero en concreto, ¿qué ventajas ofrecen las redes neuronales respecto de las técnicas estadísticas? En primer lugar, los métodos neuronales resultan relativamente fáciles de emplear, y la interpretación de sus resultados resulta asequible a muchos usuarios. Por otro lado, los modelos neuronales normalmente no parten de restricciones respecto de los datos de partida (tipo de dependencia funcional) ni suelen imponer presupuestos (como distribución gaussiana u otras).

Todavía se pueden señalar más ventajas. La respuesta de una red neuronal suele ser más rápida que la proporcionada por las técnicas estadísticas, [12], y cuando se requiere una respuesta más rápida aún (tiempo real) la red neuronal puede realizarse electrónicamente, en forma de circuitos específicos con capacidad de cálculo paralelo. Además, las redes neuronales, gracias a su posibilidad de entrenamiento en directo, pueden utilizarse para aplicaciones de control industrial, en las que los patrones van llegando uno tras otro, tarea imposible para una herramienta estadística en la que se requiere de la presencia de todos los datos simultáneamente desde el principio, [12].

Sin embargo, la ventaja más palpable quizás sea que en numerosas aplicaciones se están consiguiendo con redes neuronales cotas de error mucho menores que las proporcionadas por la Estadística. Es decir, una red neuronal no tiene porqué ser siempre la mejor solución, pero en ocasiones sí que lo es [95]. Si a alguien no le resultan suficientes estas razones, se le podría proporcionar una última motivación para el trabajo con redes neuronales, manifestada informalmente por H. White, [95], estadístico que lleva trabajando en redes neuronales muchos años: *la aplicación de las redes neuronales resulta mucho más amena y creativa que la de las técnicas estadísticas.*

A continuación vamos a ilustrar la discusión anterior con unos cuantos

ejemplos de cómo algunas técnicas clásicas de la Estadística se pueden interpretar e implementar como usos o casos particulares de ciertos métodos neuronales. Esto nos va a sugerir que, de alguna manera, la clase de métodos neuronales engloba como subconjunto a dichos métodos estadísticos tradicionales, si bien debemos tomar esta idea con precaución, pues con casi total seguridad existen ciertos métodos estadísticos más potentes que los neuronales y, por tanto, no implementables de forma directa utilizando una red neuronal.

Modelo de Regresión Lineal Estos modelos pueden ser representados mediante una red neuronal *feedforward* (hacia delante) de dos capas, denominada **Adalina**: el significado de dicho término ha cambiado ligeramente con el paso de los años; inicialmente se llamaba *Adaline Linear Neuron*, posteriormente se definió como *(ADAptive LINear Element)* [98], que posee una función de transferencia lineal o identidad.

Se sabe que la Adalina tiene la misma arquitectura esencial del modelo Perceptrón, pero con la diferencia de que para la Adalina se tiene una función de transferencia lineal, mientras que en los modelos de perceptrón se utilizan funciones tangente hiperbólica o exponenciales.

La red Adalina y su versión múltiple, *Madalina*, utilizan la *regla delta* de Widrow y Hopf, o regla del mínimo error cuadrático medio (algoritmo LMS, es decir *Least Mean Squares*, mínimos cuadrados lineales). Este algoritmo supone que la actualización de los pesos es proporcional al producto del error que la neurona comete, por su valor de entrada.

El error cometido por el modelo mide la diferencia entre el valor deseado y la salida lineal, mientras que en el modelo de perceptrón la comparación se lleva a cabo con respecto a una salida binaria (esto es debido a que, para valores nominales de operación de las neuronas del perceptrón, las salidas se saturan fácilmente a los valores extremos, que pueden ser 0 y 1, -1 y 1, etc., dependiendo de la función de activación concreta. Esta diferencia permite que los modelos Adalina/Madalina alcancen el mínimo del error de forma más sencilla que el modelo de perceptrón, así como asegurar la convergencia del proceso de entrenamiento.

La expresión de un modelo lineal y de una Adalina es la misma en la práctica, es decir, vienen dados por la ecuación siguiente:

$$y = w_0 + \sum_{i=1}^{P} w_i x_i; \;\; X = (x_1, x_2, \ldots, x_P)^T \;\; , \tag{1.12}$$

donde y es el valor de salida, X es el vector de entrada, y $(w_i)_{i=0}^{P}$ es el vector de ponderaciones o coeficientes de la combinación lineal.

Este modelo posee algunas ventajas como, por ejemplo, que no presupone aspectos como la homocedasticidad ni la ortogonalidad (que son las premisas del modelo de regresión lineal) permitiendo una mayor robustez en el proceso de estimación.

Las limitaciones que posee el modelo Adalina pueden ser solucionadas planteando una nueva topología: la red lineal adaptativa múltiple (Madalina). Esta red es similar al modelo perceptrón multicapa (MLP: *Multilayer perceptron*) y puede ser utilizada para representar modelos con regresores aparentemente no relacionados, o sea aquellos donde la relación no se puede expresar de forma analítica (o se carece de la información a priori necesaria para hacerlo, o bien obtener dicha información resulta costoso o inadecuado). La información en este caso viene generada por las correlaciones entre los términos de error; para ello se utilizan salidas retardadas como entradas en una red Adalina, obteniéndose una ecuación formada por elementos temporales de carácter lineal, AR(p), es decir, un modelo autorregresivo cuyo orden es igual al número de coeficientes lineales del modelo neuronal, suponiendo un sesgo (*bias*) nulo. La expresión de este modelo, que se muestra en la figura 1.5, sería:

$$y = w_0 + \sum_{i=1}^{P} w_i y_{t-i}; \;\; X = (y_{t-1}, y_{t-2}, \ldots, y_{t-P})^T \;\; . \tag{1.13}$$

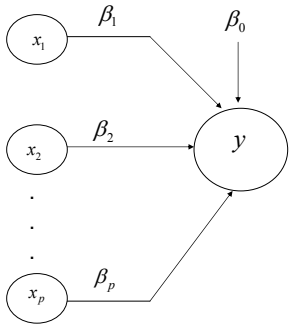


Figura 1.5: Modelo neuronal autorregresivo.

Finalmente existen otros métodos relacionados con los modelos aditivos generalizados con la misma finalidad y que son no menos importantes. Resaltamos, por su relación más cercana al caso descrito de las redes neuronales, el procedimiento adaptativo para la regresión mediante el método MARS (*Multivariate Adaptive Regression Splines* [33]).

1.2.2. Consideraciones prácticas

Limitaciones de las redes neuronales

Existen tres limitaciones fundamentales de los modelos neuronales según [63]:

1. No existe ninguna teoría formal para determinar la estructura óptima de un modelo neuronal. Hasta el momento se ha recurrido en la literatura a diversas soluciones parciales y no muy eficientes, tales como por ejemplo un mecanismo de búsqueda mediante *algoritmos genéticos*[3], [49]. La determinación del número adecuado de capas, el número de neuronas en la capa oculta, etc, se han venido decidiendo en muchos casos de manera heurística.

2. No existe un algoritmo óptimo que asegure de manera consistente el mínimo global en la superficie de error cuando esta presenta mínimos locales. Es decir, los algoritmos clásicos de búsqueda tales como la técnica de *descenso en gradiente* quedan atrapados en soluciones suboptimales representadas por dichos mínimos locales. Si bien se han propuesto mejoras a dichos algoritmos, tales como la técnica de *enfriamiento simulado* [1] (SA: *simulated annealing*), que procuran sacar a los procedimientos de búsqueda o ajuste de dichos mínimos locales, su aplicación se ve dificultada por su excesiva demanda computacional y la falta de bases teóricas más sólidas que determinen su convergencia en casos generales.

3. Las propiedades que permiten una caracterización estadística del modelo no están generalmente disponibles para las redes neuronales y, por lo tanto, no se puede llevar a cabo ninguna inferencia estadística con garantías. Además es difícil llegar a interpretar los parámetros de un modelo neuronal una vez terminado el proceso de aprendizaje.

[3]Un *algoritmo genético* es un procedimiento de búsqueda estocástica de soluciones en un espacio paramétrico de gran dimensionalidad. La búsqueda se inicia a partir de una serie de individuos iniciales (que codifican soluciones iniciales factibles o tentativas al problema de búsqueda). Los individuos se ven sometidos durante un gran número de iteraciones a operadores tales como: mutación (que somete a un individuo a un cambio pequeño en alguno de sus componentes), cruce (que da lugar a una nueva solución a partir de la mezcla de dos soluciones anteriores), etc. También, se lleva a cabo periódicamente una selección de los mejores individuos (las mejores soluciones exploradas hasta el momento) similar a un proceso de selección natural biológica. Tras un cierto tiempo de ejecución de este proceso, se confía en disponer de unas buenas soluciones al problema de partida.

Frente a todas las dificultades anteriores, existen investigadores como por ejemplo, Cheng y Titterington [7], que han realizado una labor muy importante para conectar las disciplinas de los métodos estadísticos y la tecnología de redes neuronales.

Teoría estadística del aprendizaje

En esta sección se continúa con la caracterización estadística de las redes neuronales describiendo una Teoría del Aprendizaje que trata de la cuestión fundamental de cómo controlar la habilidad de generalización de una red neuronal en términos matemáticos. La discusión es presentada en el contexto del aprendizaje supervisado. Como se mencionó en su momento, un modelo de aprendizaje supervisado consta de tres componentes interrelacionados y descritos en términos matemáticos como sigue [91, 92]:

1. **Ambiente** (o entorno). El ambiente se considera estadísticamente estacionario, y se considera que produce valores sucesivos de un vector aleatorio, $\mathbf{X}$ con una función de distribución de probabilidad fija, pero desconocida, $F_{\mathbf{X}}(\mathbf{x})$.

2. **Profesor** (o respuesta deseada). El profesor produce una respuesta deseada, d, para cada realización, $\mathbf{x}$, del vector de entrada, $\mathbf{X}$, recibido del ambiente, de acuerdo con una función de distribución condicional, $F_{D|\mathbf{X}}(d|\mathbf{x})$, que es también fija pero desconocida. Se considera que dicha respuesta deseada, d, y el vector de entrada, $\mathbf{X}$, están relacionados por la densidad, $d = f(\mathbf{x}, v)$, siendo v un término de ruido que se supone implícito al proceso ambiente-profesor.

3. **Máquina (o algoritmo) de aprendizaje**. La máquina de aprendizaje (en nuestro caso particular sería la red neuronal) es capaz de implementar una de entre un conjunto general dado de funciones de mapeo (determinísticas) de entrada-salida, $\mathcal{F} = \{\hat{F}(\cdot, \mathbf{w}) : \mathbf{w} \in \mathcal{W}\}$, que notaremos por $y = y_{\mathbf{w}}(\mathbf{x}) = \hat{F}(\mathbf{x}, \mathbf{w})$, evaluada sobre el propio dato, $\mathbf{X}$, emitido por el ambiente, donde $\mathbf{w}$ denota un cierto vector de parámetros interno a la máquina o algoritmo, moviéndose en un determinado espacio paramétrico, $\mathcal{W}$. El valor actual de dicho vector, $\mathbf{w}$, particulariza la clase de funciones que implementa la red, obteniéndose una función concreta que usará la red para aproximar la función real de entrada-salida del contexto ambiente-profesor.

El problema del aprendizaje supervisado es seleccionar la función particular, $\hat{F} \in \mathcal{F}$, que mejor aproxima la función de respuesta deseada, donde el

término mejor se define en un cierto sentido estadístico. La propia selección se debe basar en un conjunto de P ejemplos de entrenamiento independientes e idénticamente distribuidos, $\Psi = \{(\mathbf{x}_i, d_i)\}_{i=1}^{P}$.

Cada par de ejemplos es presentado a (o extraído por) la máquina de aprendizaje a partir del conjunto Ψ de acuerdo con la función de distribución conjunta, $F_{\mathbf{X},D}(\mathbf{x}, d) = F_{\mathbf{X}}(\mathbf{x}) \cdot F_{D|\mathbf{X}}(d|\mathbf{x})$, que, como las otras funciones de distribución, es también fija pero desconocida.

La viabilidad del aprendizaje supervisado depende de este tema: ¿los ejemplos de entrenamiento, $\{(\mathbf{x}_i, d)\}$, contienen información suficiente para construir una máquina de aprendizaje capaz de tener buen rendimiento a la hora de generalizar? Una respuesta para este tema fundamental está en la utilización de herramientas teóricas que se encuentren a la altura de la formulación y tratamiento de un problema tan general y a la vez tan específico y dependiente de la arquitectura concreta de aprendizaje empleada.

Específicamente, se parte (como hemos señalado) de la consideración del problema del aprendizaje supervisado como un problema de aproximación, que implica encontrar la función, $\hat{F}(\mathbf{x}, \mathbf{w})$, que sea la mejor aproximación posible para la función deseada, $d, d = f(\mathbf{x})$, dentro de la clase de funciones, $\mathcal{F}$, implementables usando dicha red. Se derivan posteriormente cotas matemáticas sobre la capacidad de un conjunto de funciones implementadas por una clase general de sistemas o redes, y se atribuye dicha capacidad, C, como una medida general de la potencia de aproximación del modelo considerado.

Al mismo tiempo, en estos estudios teóricos avanzados se derivan otras ciertas cotas referidas al *ratio* o velocidad esperada de convergencia del proceso de aprendizaje, la expresión y derivación de las cuales excede con mucho los propósitos de esta memoria; el lector interesado puede recurrir a los textos adecuados sobre la teoría del aprendizaje formulada por [89], o la teoría de la dimensión VC de Vapnik-Chervonenkis [92], etc., las cuales no dejan en todo caso de ser especializaciones sumamente elaboradas de conceptos teóricos ya considerados con anterioridad en la literatura existente sobre la llamada *teoría de procesos empíricos* [94] y que incluso encuentran acomodo en la teoría estadística de aproximaciones estocásticas [47].

El aprendizaje visto como un problema de reconstrucción de una hipersuperficie

En el campo de las redes neuronales, se menciona de forma invariable, directa o indirectamente [61, 78, 35], el concepto de *aprendizaje* [50, 14]. ¿Qué se entiende por capacidad de una red neuronal (o de un sistema en general) para aprender?

Una teoría profunda sobre la naturaleza del aprendizaje es una tarea de

veras pretenciosa y nos desviaría en extremo del objetivo de este trabajo. No cuesta, sin embargo, intentar delimitar el concepto en el contexto que nos ocupa, aunque sea de forma breve o esquemática.

Como punto de partida vamos a considerar un experimento de aprendizaje y condicionamiento animal bastante similar a otros bien conocidos; sin embargo, en su discusión aparecen algunos conceptos e ideas básicos que constituyen una base de conceptos imprescindible para cualquier experimento medianamente serio que involucre teorías sobre el aprendizaje. Este puede ser indistintamente de tipo animal, humano, estadístico o, como el caso que nos ocupa, un híbrido entre la teoría del aprendizaje de máquinas —disciplina tradicionalmente entendida como una ramificación más de los estudios teóricos generales sobre la llamada Inteligencia Artificial (IA)— y la teoría estadística del aprendizaje que se ha venido postulando en nuestras disquisiciones anteriores[4].

La experiencia a que nos referimos fue desarrollada por Nicolelis y Chapin [64], ambos neurocirujanos. Utilizaron técnicas psicológicas de condicionamiento operativo desarrolladas durante décadas de investigación del aprendizaje de animales para adquirir algún control de las señales del cerebro. Los ratones aprenden a bajar una palanca que, a su vez, activa un brazo mecánico que les da una recompensa (agua), y cada vez que el animal baja la palanca, los electrodos implantados en su cerebro permiten a los investigadores observar y registrar las señales nerviosas que acompañan dicha actividad.

En la siguiente fase los científicos simplemente desligan la barra de la

[4]Nótese que el concepto de *Machine Learning* (aprendizaje de máquina) está inicialmente más asociado a la discriminación o aprendizaje de conceptos que al análisis o procesamiento de señales o datos. Por decirlo de otra manera, el campo del *machine learning theory* se sitúa en un nivel más alto que la teoría estadística del aprendizaje (*statistical learning theory*) que hemos citado. Es importante tener en cuenta la distinción existente, por mucho que la nomenclatura similar lleve a confusión y por mucho que, para una gran mayoría de personas, la palabra *machine* tenga conotaciones más físicas y tangibles (computacionales) que la palabra *statistical*. Por tanto, en definitiva, nuestra aproximación es más estadística, más cercana al dato que al conocimiento implícito que ese dato deja entrever y, en este sentido, nuestro enfoque —y cualquier enfoque contemporáneo que use las redes neuronales en entornos híbridos de procesado de señal, predicción o comunicaciones— se aleja bastante del espíritu abstracto de la Inteligencia Artificial (la cada vez menos práctica imagen de la *máquina que aprende conceptos*) para adentrarse en otro campo más específico (y creemos que con resultados bastante más aplicables e implementación más directa) que preferimos denotar —siguiendo la notación de Robert J. Marks II en [57]— como *Inteligencia Computacional* (término también sustituido en algunos ámbitos por el de *Soft Computing*) y que se ocupa, básicamente, del procesamiento, control y predicción de datos y/o señales mediante técnicas de Redes Neuronales, Sistemas Difusos (*Fuzzy Systems* [11]), Algoritmos Genéticos, o cualquier combinación de dichos tres paradigmas.

palanca por un período de tiempo dado. En un primer momento, esto provoca la frustración en el ratón al percibir que el movimiento, aprendido con tanto esfuerzo, ya no le proporciona la recompensa. Sin embargo, tras alguna insistencia del animal, el brazo de improviso vuelve a descender y libera la recompensa. Después de algún tiempo así, el ratón percibe que no precisa de hecho empujar la barra, sino que basta con mirarla e *imaginar* su pata delantera empujándola. De esa forma, sus neuronas generaban el mismo patrón de señales nerviosas y la palanca quedaría activada. La conclusión extraordinaria es que el ratón había aprendido a mover la palanca con la fuerza de la mente [2].

Esta interesante experiencia ilustra dos categorías de aprendizaje diferentes. La primera, más obvia, se refiere a los movimientos que debe desempeñar el ratón: en primer lugar, asociando el movimiento de su pata delantera con el premio pero, posteriormente, percibiendo que un simple pensamiento daba lugar al mismo beneficio. La segunda categoría de aprendizaje es un poco más sutil, pero esencial para el éxito de la experiencia: la detección del patrón de señales por las 46 neuronas cuando el ratón hacía el movimiento deseado.

Aunque diferentes, ambos ejemplos de aprendizaje encierran un factor común: algún tipo de asociación. En el primer caso, el animal aprende a asociar un movimiento –o pensamiento– con la recompensa. En el segundo caso, un conjunto de 46 niveles potenciales de acción específica se asocia con una salida análoga, sin existir un patrón motor asociado (o mejor dicho, un conjunto de los mismos).

Dicho de otra forma, la distinción está en que en el primer caso, los patrones específicos que se iban formando en el cerebro del ratón podían ser muy variados, aún cuando obviamente tendrían que compartir bastantes características entre sí. Esto se debía fundamentalmente a las diferentes posiciones del ratón con respecto a la palanca, lo que inevitablemente conducía a la activación de diferentes grupos de músculos para cada intento de activarla. Sin embargo, en la segunda modalidad de aprendizaje sus neuronas aprenden de hecho a generar un patrón mucho más específico y determinado, que es el que (a través del mecanismo de respuesta) se había forzado a ajustar usando diversas calibraciones por medio de resistencias electrónicas.

Esto contiene un cierto paralelismo con la distinción entre aprendizaje con refuerzo (donde se obtiene o se niega una recompensa en el proceso) y aprendizaje auto-organizado (donde el propio ratón consigue modificar el mecanismo de activación de patrones en sus neuronas). No hay varios patrones de respuesta idóneos, ni función objetivo, sino que el objetivo es que las neuronas lleguen a establecerse en dicho estado peculiar del sistema.

Lo anterior significa que, en ciertos problemas de aprendizaje, un conocimiento limitado almacenado en forma de una tabla de consulta (*look-up*

table) puede no ser muy útil o, si lo es, tendría que ser tan exhaustivo que sería poco implementable en un algoritmo o sistema que operase en tiempo real. En nuestro ejemplo, codificar todos los posibles patrones nerviosos asociados a todos los movimientos musculares que en algún momento dieron lugar a la generación del refuerzo, no sería práctico y diría poco en términos de aprendizaje, limitándose a reflejar múltiples asociaciones puntuales que, externamente, se podrían considerar hasta cierto punto arbitrarias [2].

El mundo en que vivimos es, considerando el nivel de abstracción adecuado, redundante [70]. Eso significa que existen ciertos patrones que se repiten de forma consistente en el tiempo, presentando cierta coherencia en el espacio de datos. Si no fuera de esa forma, nos sería imposible extraer cualquier conocimiento de la experiencia. Como ejemplo, [70] cita el caso de una guía telefónica: se puede almacenar una cantidad infinitamente grande de números de teléfonos, pero eso de nada nos ayudaría a la hora de estimar el teléfono de una persona que no estuviese en la lista. Podemos aprender de memoria (pero sin poder extraer ninguna regla o fórmula cerrada en el proceso) la guía de teléfonos; esto es así por la sencilla razón de que no presenta regularidades (en principio) entre los apellidos de una persona y los dígitos concretos que se le asocian en la guía.

Por fortuna, los eventos o datos del mundo real no son totalmente locales: patrones parecidos tienden a generar respuestas parecidas. El espacio de los datos de la vida real presenta entonces ciertos rasgos de continuidad. De cierta forma, se puede decir que es posible extraer un mapa, de cierta suavidad, que describa una porción de la realidad, y dicha suavidad es precisamente la base que permite, al menos localmente, la *generalización*.

Tal vez por eso ahora podemos entender que el ratón de la experiencia antes descrita tenía inicialmente dificultades para lograr obtener el el agua con el movimiento de la pata. Como el mapeo de su patrón neuronal asociado a los movimientos estaba siendo hecho de manera artificial (a través del sistema programado con resistencias que se estaba usando) se puede suponer que no existía flexibilidad suficiente para la detección de movimientos muy diferentes de los originales. En el inicio, como el objetivo era empujar la palanca y no repetir el patrón mental, el ratón pudo ir intentando ocasionalmente movimientos alternativos. Después de algún tiempo percibió que la manera en que se empujaba la palanca tan solo *pensándolo*, sin ser una copia perfecta de ninguna de las alternativas exitosas anteriores, era fundamental para el éxito de la operación.

Evidentemente, esto son suposiciones preliminares que requerirían un estudio adicional y detallado. El objetivo aquí no es discutir a fondo la experiencia descrita, sino usarla como forma de ilustración del asunto en cuestión. Para una discusión mas detallada y menciones de otras experiencias del mis-

mo tipo, se aconseja consultar la fuente original [64] y las referencias en ella contenidas.

Tal vez este momento de nuestra discusión sea el más adecuado para relacionar algunas ideas presentadas con conceptos matemáticos. Hasta ahora, el termino mapeo ha sido utilizado para describir las asociaciones incluídas en el proceso de aprendizaje. Matemáticamente, la idea de mapeo puede ser modelada a través del concepto de función. En el caso del experimento descrito, la relación entre el estado de activación de las 46 neuronas del ratón y la salida analógica podría ser descrita por una función, f, de la siguiente forma:

$$\begin{aligned} f : \mathbb{R}^{46} \rightarrow \mathbb{R} \ ; \\ f : \mathbf{x} \mapsto y \ , \end{aligned} \tag{1.14}$$

donde los valores de y representarían los movimientos del brazo del animal, de los cuales un cierto valor específico es el que se considera que activa la palanca para liberar el agua.

Se puede considerar la función, f, como dando lugar a una hipersuperficie en el espacio de 46 variables más la variable respuesta; es decir,[4, 35],

$$(\mathbf{x}, y) \in \Lambda \equiv \mathbb{R}^{46}.$$

Dicha hipersuperficie puede ser vista de la forma usual, como una gráfica, A, multidimensional de la salida en función de las 46 entradas. La superficie se determina a partir de algunos puntos de ejemplo (obtenidos en este caso mediante registro, por parte de los científicos, de los valores observados de activación neuronal, y la desviación analógica de la palanca que el ratón lleva a cabo).

La motivación de esa postura es la consideración de que existe (en el mundo real) cierta hipersuperficie, probablemente suave, que describe el fenómeno real perfectamente. Los patrones de ejemplo-respuesta son puntos obtenidos experimentalmente, y por tanto contaminados con ruido, que pertenecen a Λ.

De esa forma, el aprendizaje es visto como un problema de reconstrucción de una hipersuperficie, dado un conjunto de puntos con ruido que pueden ser obtenidos de forma dispersa [35]. Pero esa dispersión y ese ruido influyen profundamente en la definición del problema y en la naturaleza de las técnicas que deben considerarse para su adecuada resolución. En este sentido, la *Teoría de la Regularización* (también considerada la teoría de los problemas mal definidos, *ill-posed* en inglés) ofrece una metodología general que tendremos en cuenta en nuestro caso particular de las redes neuronales como

estructuras con procesos de aprendizaje asociados en el sentido que estamos mencionando.

Comparación entre algoritmos neuronales y métodos estadísticos de ajuste

Desde un punto de vista estadístico, muchos de los problemas que se intentan resolver con *backpropagation*[5] (o con redes neuronales en general) entran en la categoría de los denominados *problemas mal condicionados* o mal formulados. Se entiende por tal tipo de problemas aquellos en los que el espacio de trabajo es tan amplio y los datos disponibles tan escasos, que resulta *a priori* difícil encontrar la red neuronal que los ajuste correctamente, puesto que la información contenida en los datos de entrenamiento no es suficiente para determinar unívocamente el mapeo, de manera que las posibles soluciones que en principio permiten ajustar los datos resultan ser virtualmente infinitas [35].

Hablando en términos estadísticos, las redes neuronales son estimadores no paramétricos que realizan estimaciones denominadas de modelo libre. Por ejemplo, el método convencional de ajuste a una línea recta mediante mínimos cuadrados sería un estimador paramétrico, pues se impone al problema un determinado modelo de partida, la línea recta, cuyos parámetros se deben ajustar según las muestras disponibles.

A diferencia de los paramétricos, los métodos de redes neuronales como el algoritmo *backpropagation*, los algoritmos de RBF, etc., constituyen estimadores de modelo libre, pues no se impone ninguna forma funcional de partida concreta (de entre la clase de funciones particularmente realizable por la red). Por ejemplo, con las funciones gaussianas de una red de RBF se puede interpolar cualquier función continua hasta cumplir una condición determinada de precisión, pero *a priori* no se distingue una clase particular de funciones implementables (con tal de que se comporten razonablemente bien).

Para desarrollar una mayor comprensión del problema del ajuste excesivo y de cómo tratarlo, retornamos al punto de vista de que una red neuronal entrenada para recuperar un patrón de salida cuando se presenta un patrón de entrada es equivalente a encontrar una hipersuperficie (un mapeo multidi-

[5]Recordamos que se usa el término *backpropagation* para referirse al algoritmo de entrenamiento más comúnmente usado a la hora de ajustar redes neuronales del tipo perceptrón mono o multicapa, siendo un algoritmo basado primordialmente en la retropropagación del error (desde la capa final hacia las capas previas de la red) mediante el empleo de expresiones numéricas relativas a las derivadas parciales de los términos que involucran error neuronal con respecto a los parámetros ajustables de la red.

mensional) que define la salida en términos de las entradas. En otras palabras, el aprendizaje es visto como un problema de reconstrucción de una hipersuperficie, dado un conjunto de puntos de datos que pueden estar sujetos a ruido [35].

Capítulo 2

Funciones de base radial

2.1. Funciones de base radial de tipo elíptico

En las funciones de base radial RBF (*Radial Basis Function*) típicas, interviene la distancia entre el vector de entrada actual, $\mathbf{x}$, y el centro, $\mathbf{c}_i$, de la neurona considerada:

$$r = \|\mathbf{x} - \mathbf{c}_i\| = \sqrt{(\mathbf{x} - \mathbf{c}_i)^T \cdot (\mathbf{x} - \mathbf{c}_i)} \tag{2.1}$$

Si en vez de la norma euclídea en la ecuación (2.1) se utiliza la norma de Mahalonobis, la distancia se transforma según la norma de la matriz Σ.

$$r = \|\mathbf{x} - \mathbf{c}_i\|_{\underline{\Sigma}_i} = \sqrt{(\mathbf{x} - \mathbf{c}_i)^T \cdot \underline{\Sigma}_i \cdot (\mathbf{x} - \mathbf{c}_i)} \ , \tag{2.2}$$

donde el conjunto de parámetros contiene ahora además la norma de la matriz Σ. Esta matriz escala y rota los ejes de coordenadas. Para el caso especial donde se tiene la matriz con la diagonal formada por los inversos de los σ_i al cuadrado, la matriz Σ coincide con la identidad ($\Sigma = \mathbf{I}$) y la norma de Mahalonobis es igual a la norma euclídea. Para

$$\underline{\Sigma} = \text{diag}\left(\frac{1}{\sigma_1^2}, \frac{1}{\sigma_2^2}, \cdots, \frac{1}{\sigma_p^2}\right) \ , \tag{2.3}$$

donde p denota la dimensión del espacio de entradas, X. En el caso general, se incluyen en la matriz términos tanto de escalado como de rotación de los ejes de entrada.

La figura 2.1 resume estas medidas de la distancia. Por eso la función se llama función de base radial. A pesar del hecho de que no es radial (a veces ni siquiera simétrica) con respecto a los datos de entrada, sí que se puede

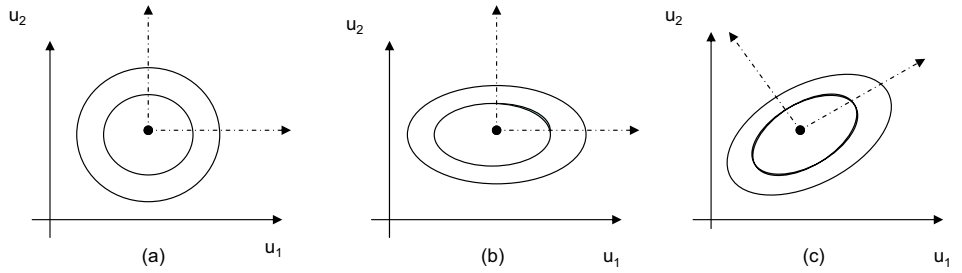


Figura 2.1: Diferentes tipos de normas

considerar como radial con respecto a los ejes de entrada una vez hayan sido convenientemente transformados (desplazados y/o rotados).

Arquitectura de una RBF

En la figura 2.2 se muestra la arquitectura de una función de base radial. Las redes de neuronas de base radial son redes con conexiones hacia adelante, como se observarse en la figura 2.2, y estas conexiones se dirigen siempre de una capa a siguiente capa.

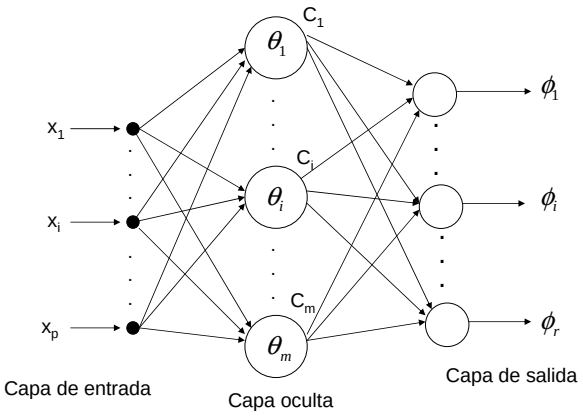


Figura 2.2: La i-ésima neurona oculta de una red de RBF.

Las redes de neuronas de base radial definen una relación no lineal entre las variables de entradas y las variables de salida de la red, propagando hacia la salida las señales o muestras recibidas en la entrada. A continuación, se presentan las expresiones para calcular las activaciones de las neuronas de las redes de base radial.

En la primera fase, se determina la distancia del vector de entradas, $\mathbf{x} = (x_1, x_2, \ldots, x_p)^T$, al vector de centros, $\mathbf{c}_i = (c_i^{(1)}, c_i^{(2)}, \ldots, c_i^{(m)})^T$, y también se aplica la matriz de norma Σ. A continuación esta distancia, r, (un escalar) se ve transformada por una función de activación, $\theta(\cdot)$, no lineal.

A menudo la matriz Σ puede ser diagonal, conteniendo los inversos de los cuadrados de los radios para cada dimensión de entrada. En ese caso particular, la función de distancia para la RBF se calcularía como

$$\|\mathbf{x} - \mathbf{c}_i\|_{\Sigma_i} = \sqrt{\sum_{j=1}^{p} \left(\frac{x_j - c_i^{(j)}}{\sigma_{ij}} \right)^2} = \sqrt{\left(\frac{x_1 - c_i^{(1)}}{\sigma_{i1}} \right)^2 + \cdots + \left(\frac{x_p - c_i^{(p)}}{\sigma_{ip}} \right)^2} \ , \tag{2.4}$$

ya que, en dicho caso,

$$\Sigma_i = \text{diag} \left(\frac{1}{\sigma_{i1}^2}, \frac{1}{\sigma_{i2}^2}, \cdots, \frac{1}{\sigma_{ip}^2} \right) \ . \tag{2.5}$$

A continuación comentamos el efecto de algunas matrices de norma en la forma de la función de la base. Así, para anchuras idénticas cada dimensión conduce a una verdadera función de base radial con contornos de círculo, conforme a la ecuación

$$\Sigma = \begin{pmatrix} 1/\sigma_{i1} & 0 \\ 0 & 1/\sigma_{i1} \end{pmatrix} \ . \tag{2.6}$$

Por otro lado, considerando diversas anchuras para cada dimensión tendremos una función radial simétrica con contornos elípticos, es decir,

$$\Sigma = \begin{pmatrix} 1/\sigma_{i2} & 0 \\ 0 & 1/\sigma_{i1} \end{pmatrix} \ . \tag{2.7}$$

En la ecuación siguiente, por su parte, se ilustra el uso de una matriz completa, que permite implementar una rotación de las funciones de base:

$$\Sigma = \begin{pmatrix} 1/\sigma_{i2} & 1/\sigma_{12} \\ 1/\sigma_{21} & 1/\sigma_{i1} \end{pmatrix} \ . \tag{2.8}$$

Haciendo cálculos y simplificando en la ecuación (2.5) para el caso $\sigma_{i2} = \sigma_{i1}$ se obtiene:

$$\theta_i(x) = \exp \left(-\frac{1}{2} \frac{\|x - c_i\|^2}{\sigma_i^2} \right) \ . \tag{2.9}$$

Este resultado muestra que la función aún sigue dependiendo del índice de cada neurona, o sea, cada neurona tiene sus radios particulares, σ_{i2} y σ_{i1}, sólo que en este caso los consideramos idénticos.

Una gráfica ilustrativa de las ecuaciones (2.6), (2.7) y (2.8) se observa en la figura 2.1.

También se observa que cada neurona posee su propio centro, $\mathbf{c}_i$. En la práctica, si se tienen miles de datos, no es razonable poner una neurona dedicada y centrada en cada uno. Esto es debido a que si se permiten demasiadas neuronas (p.ej. número de neuronas igual al número de ejemplos) se necesita invertir una matriz de dimensión igual a dicho número de ejemplos, que en la práctica suele ser elevado, lo que resulta prohibitivo.

Por otro lado, si el modelo es poco particular (p.ej., compartiendo todas las neuronas un radio común) existe un problema de poca flexibilidad del modelo, y no se pueden ajustar individualmente los radios, σ_i. Por ello, en la literatura se propuso el modelo general de radios individuales que, aunque implica un mayor número de parámetros a determinar, permite un ajuste más preciso a los datos. En este trabajo se considerarán radios fijos. El valor del radio es un valor óptimo encontrado con el objetivo de realizar la reducción de neuronas RBF.

Interpolación

Para resolver el problema de separación no lineal, es conveniente transformarlo en uno lineal aunque sea de mayor dimensión:

$$\Re^n \xrightarrow{\theta(x)} \text{(no lineal)} \quad \Re^m \xrightarrow{w} \text{(lineal)} \quad \Re \tag{2.10}$$

Encontramos por tanto dos fases:

1. **Entrenamiento**. Proceso de ajuste de los puntos dados por la curva.
2. **Interpolación**. Interpolar nuevos datos.

El problema de interpolación puede enunciarse como que dado un conjunto de n puntos, $x_i \in \Re^p, i = 1, 2, \ldots, n$, y otro conjunto asociado de n números reales, $d_i, i = 1, 2, \ldots, n$, encontrar la función $\phi : \Re^p \to \Re$, tal que $\phi(x_i) = d_i, i = 1, 2, \ldots, n$.

La técnica de base radial para la interpolación consiste en seleccionar ϕ como se indica en la ecuación 2.11, donde $\phi(\cdot)$ es un conjunto de n funciones no lineales, conocidas como bases radiales y $\|\cdot\|$ es la norma, usualmente la euclidiana (radio).

Con los elementos $\theta_{ij}, i, j = 1, \ldots, n$, se formará la matriz θ la cual se denomina matriz de interpolación; y que se puede reescribir como 1.1. Suponiendo que θ es no singular, es posible resolver como en 1.2. Sin embargo, no es posible asegurar que la matriz de interpolación, θ, sea no singular en el caso general. La solución la da el siguiente teorema.

Teorema 2.1.1 *Michelli, 1986. Sea $x_1, x_2, \ldots, x_p$ un conjunto de puntos distintos en R^n. Si $\phi \in R^{p \times p}$ está formada como en 2.11 y $\theta(\cdot)$ pertenece a la clase de funciones de base radial, entonces $\theta(\cdot)$ es no singular.*

Activaciones de las neuronas de la red de base radial

Dada una red de neuronas de base radial con p neuronas en la capa de entrada, m neuronas en la capa oculta y r neuronas en la capa de salida, las activaciones de las neuronas de salida para el patrón de entrada n, $X(n) = (x_1(n), x_2(n), \ldots, x_p(n))$, son denotadas como $\phi_r(n)$. Así, la técnica de las RBF consiste en escoger una función de aproximación, $\bar{\phi}$, que tiene la siguiente forma:

$$\phi_r(n) = \sum_{i=1}^{m} w_{ir}\theta_i(n) + u_r \tag{2.11}$$

donde w_{ir} representa el peso de la conexión de la neurona oculta, i, a la neurona de salida, r; u_r es el umbral de la neurona de salida r, y $\phi_i(n)$ son las activaciones de las neuronas ocultas para el patrón de entrada, $X(n)$. Se observa en la ecuación 2.11 que las neuronas de salida de la red utilizan la función de activación identidad, realizando una transformación lineal de las activaciones de todas las neuronas ocultas [44].

La función de base radial, ϕ, puede adoptar diferentes formas y expresiones, entre otras se usarán en este trabajo:

Función lineal:

$$\theta_i(\mathbf{x}) = \|\mathbf{x} - \mathbf{c}_i\|^2 \text{ para algún } c > 0. \tag{2.12}$$

Función gaussiana:

$$\theta_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{2\sigma^2}\right) \text{ para algún } c > 0 \text{ y } \sigma > 0. \tag{2.13}$$

donde σ es un parámetro que controla las propiedades de suavidad de la interpolación de la función.

Función multicuadrática:

$$\theta_i(\mathbf{x}) = \sqrt{\|\mathbf{x} - \mathbf{c}_i\|^2 + \sigma^2} \text{ para algún } c > 0 \text{ y } \sigma > 0. \tag{2.14}$$

Función multicuadrática inversa:

$$\theta_i(\mathbf{x}) = \frac{1}{\sqrt{\|\mathbf{x} - \mathbf{c_i}\|^2 + \sigma^2}} \text{ para algún } c > 0 \text{ y } \sigma > 0. \tag{2.15}$$

En todos estos casos, $\mathbf{c}_i$ representa el centro de la función de base radial y σ_i es la anchura. Este parámetro, σ_i, puede ser interpretado como un factor de escala para la distancia, $\|\mathbf{x} - \mathbf{c_i}\|^2 + \sigma^2$. En el caso de la función gaussiana, por ejemplo, el valor de $\theta_i(\mathbf{x})$ decrece más rápidamente cuando $\sigma_i \to 0$. La definición de estas anchuras tiene un fuerte impacto sobre las características de la función de aproximación [28, 6, 67].

La función gaussiana y la multicuadrática inversa son funciones locales, o sea, dan lugar a una respuesta más significativa cuanto más nos acerquemos al centro correspondiente, $\mathbf{c_i}$. La función multicuadrática, a su vez, es global, pudiendo tomar $\theta_i(\mathbf{x})$ valores arbitrariamente grandes cuando la distancia al centro tiende a infinito.

Las funciones locales, especialmente la gaussiana, son más comúnmente usadas que las que presentan respuestas globales [68]. Una característica que las torna particularmente atrayentes es su mayor plausibilidad biológica. En contrapartida, algunos resultados citados en [35] indican que las funciones que tienden a infinito pueden en la práctica ser tomadas como acotadas a la hora de aproximar un mapeo de entrada-salida suave con mayor precisión que con las funciones locales.

Para el problema de interpolación estricta, la superficie de interpolación es obligada a pasar por todos los puntos dados. La generalización significa, en ese caso, interpolar la superficie en las regiones donde no hay ejemplos disponibles [4]. Como es bien observado en [70], es posible estructurar casi todos los esquemas de aproximación como algún tipo de red que puede ser considerada una red neuronal. Las redes neuronales, en definitiva, pueden ser interpretadas como una solución gráfica para una gran clase de algoritmos. En lo que sigue, se considerará a la arquitectura de red RBF como un método de implementación de la teoría de regularización.

2.2. Teoría de la regularización

Para entender el concepto de regularización usaremos una función ficticia utilizada en [2]. Esta función presenta una estructura lineal simple. Considere la función $z_t = a_o + a_1 t + a_2 t^2$. Suponga que para esta función se tiene

una gráfica como la mostrada en la figura 2.2. En esta situación hipotética, suponga también que los únicos datos de que se dispone son 50 puntos de observaciones con ruido. El problema puede ser formulado de la siguiente manera: dado un conjunto de puntos, $P = (t_i, z_{t_i}) \in \Re \times \Re, i = 1, 2, ..., 50$, se debe encontrar la función, $\bar{\phi}$, que mejor se aproxime a la función original, z_t, generadora de los puntos (es importante observar que la función, z_t, fue escogida de forma que facilitase la exposición, y todas las observaciones pueden ser fácilmente extendidas para el caso de funciones del tipo $\Re^n \longrightarrow \Re^m$.

Se puede entender aquí que se trata de un problema inverso. El problema está mal formulado, porque los datos disponibles no son suficientes para que la función considerada sea reconstruida de manera única. Otro problema es que no existe necesariamente una salida distinta para cada entrada, luego se entiende que está contaminada con ruido.

Para lidiar con problemas mal formulados, [88] ha propuesto una técnica conocida como regularización. La idea de la regularización es intentar incorporar alguna información previa a la solución del problema. Otras restricciones más fuertes pueden ser consideradas, como por ejemplo: que la función sea lineal, estar restringida a un determinado intervalo o ser invariable en relación con algún grupo de transformaciones. Evidentemente, se debe tener en consideración toda la información de que se tenga conocimiento a priori.

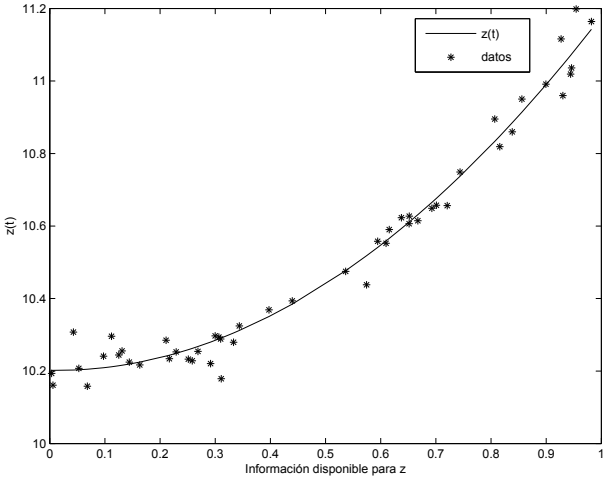


Figura 2.3: Información disponible para z

.

Básicamente, la teoría de Tikhonov considera dos términos:

1. **Término de error sobre los patrones**. Este primer término, representado por $\varepsilon_c(\phi)$, mide la totalidad del error entre las respuestas

deseadas, z_{t_i}, y las respuestas obtenidas, ϕ_{t_i}. Específicamente, se define:

$$\varepsilon(\phi) = \frac{1}{2}\sum_{i=1}^{50}\left(z(t_i) - \phi(t_i)\right)^2 \quad . \tag{2.16}$$

2. **Término de regularización**. Este segundo término, representado por $\varepsilon_c(\phi)$, depende de las propiedades geométricas de la función aproximativa, $\phi(t_i)$. Específicamente, se puede escribir:

$$\varepsilon_c(\phi) = \frac{1}{2}\|D\phi\|^2 \quad . \tag{2.17}$$

donde D es un operador diferencial lineal. La información previa sobre la forma de la solución se incorpora en ese operador.

El problema pasa a ser, entonces, encontrar la función, $\bar{z}$, que minimice el funcional de Tikhonov:

$$\varepsilon(\phi) = \varepsilon_z(\phi) + \lambda\varepsilon_c(\phi) \quad . \tag{2.18}$$

En nuestro ejemplo se tiene:

$$\zeta_{\text{Tikhonov}} = \varepsilon(\phi) = \sum_{i=1}^{50}\left[z(t_i) - \phi(t_i)\right]^2 + \lambda\|D\phi\|^2\big] \quad . \tag{2.19}$$

donde λ es el parámetro de regularización. Este parámetro de regularización controla el compromiso entre el grado de suavidad de la solución, ϕ, y su distancia a los puntos datos. Un valor pequeño de λ implica que los ajustes a los puntos pueden ser muy precisos, sin que eso genere un penalización muy grave. Si el valor de λ es muy grande, el ajuste debe ser sacrificado en detrimento de una función más simple.

Cuando el valor de λ es sobreestimado, el resultado es una función que realiza un ajuste poco preciso a los datos conocidos. Por otro lado, cuando se presenta la situación opuesta, aunque la función de aproximación pasa con precisión por todos los puntos dados, esta no daría lugar a una buena generalización en algunas regiones del intervalo. Esta situación se conoce en la literatura de redes neuronales como **exceso de ajuste** (*overfitting*) ver figura 4.8. Eso ocurre cuando el modelo es excesivamente sensible a las particularidades del conjunto concreto de datos considerado (es decir, la red termina aprendiendo el ruido existente en los datos, el cual no es parte del modelo real).

Para valores de $\lambda = 0{,}1$ y radio $r = 0{,}01$ el ajuste de esta serie ficticia se muestra en la figura 2.4. Para un valor mayor de λ se penaliza la serie más fuertemente. Para este ejemplo ficticio se tienen valores muy elevados para ambos casos, pero con la regularización se suelen obtener errores menores.

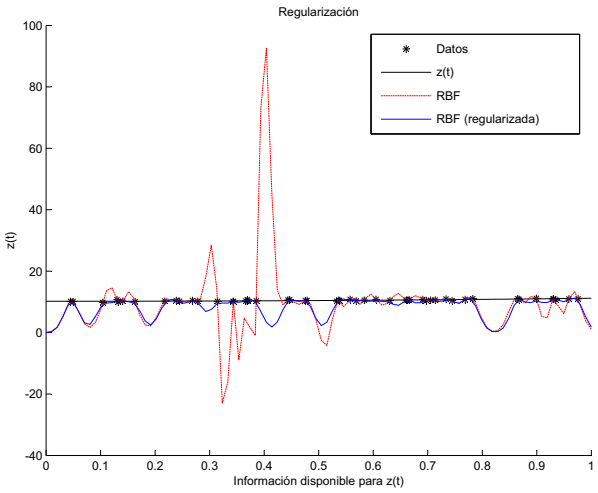


Figura 2.4: Ajuste de la serie regularizada

Se puede interpretar por otro lado este valor óptimo de λ^* como un indicador de la suficiencia del conjunto de datos empleados como ejemplos. En el caso extremo en que $\lambda^* = 0$, se podría afirmar que se trata de un problema no restringido y que la solución estaría totalmente determinada por los datos. El otro caso límite ocurriría cuando $\lambda^* \to \infty$, lo que implicaría que la información previa incorporada por el operador ya sería suficiente para resolver el problema, lo que equivale a decir que los datos no contendrían ninguna información que ayudara a su solución.

2.2.1. Ajuste lineal en las redes de RBF

Si se supone que tanto el número de funciones de base radial como todos sus centros y anchuras (o radios) se mantienen fijos durante el proceso de entrenamiento (o dichos parámetros vienen predefinidos o preentrenados), la salida de la RBF puede ser vista como un modelo lineal simple [68]. Consecuentemente el entrenamiento de la salida queda reducido a la determinación de los coeficientes asociados a cada neurona, para combinarse en la salida.

Se pueden usar aquí las técnicas usuales de ajuste de modelos lineales: en particular, se podría formular en principio como la inversión de una matriz.

La única discusión adicional sería la asociada a la estimación de un valor razonable para el parámetro de regularización, λ (si es que se desea emplear un enfoque regularizado).

Entrenamiento semisupervisado o en dos fases

Al utilizar funciones locales, la arquitectura de las RBF posibilita una forma de aprendizaje híbrida que presenta muchos atractivos. La idea principal de esta estrategia es dividir el entrenamiento en dos fases: primeramente aprendizaje no supervisado o auto-organizado, cuyo propósito es estimar localizaciones fijas adecuadas para los centros y radios (es decir, los parámetros de las funciones de base radial en la capa oculta), y posteriormente fase de aprendizaje supervisado que se realiza estimando los pesos lineales de la neurona de salida global.

En la fase auto-organizada, por tanto, se determinan las posiciones de los centros, $\mathbf{c}_i$, y las anchuras o radios, σ_i, de las RBF.

Con la capa oculta ya totalmente definida, se puede calcular fácilmente el valor de los coeficientes w_i a través de la inversión de una matriz. Si existieran problemas numéricos para ello, siempre se puede regularizar el problema, lo que equivaldría al método estadístico de *ridge regression*.

Otra opción para evitar los problemas numéricos que pueden en determinados casos impedir la inversión, es usar la pseudoinversa de la correspondiente matriz, θ que se obtendría a través de la siguiente expresión:

$$\theta^{+} = (\theta^{T}\theta)^{-1}\theta^{T} \quad . \tag{2.20}$$

Para minimizar los efectos de un posible mal condicionamiento de θ también puede ser recomendable utilizar para la inversión de esa matriz el método conocido como descomposición de valores singulares (SVD) [73, 29].

Ajuste de centros y radios

A continuación ofrecemos diversas alternativas para llevar a cabo la primera fase mencionada, es decir, el cómputo de los centros y radios de las neuronas en RBF.

Selección aleatoria. El enfoque más simple para definir la posición de los centros es simplemente escoger al azar algunos puntos de la muestra de entrenamiento para cumplir ese papel. Esta es una aproximación sensata, suponiendo que los datos de entrenamiento estén distribuidos de una forma representativa para el problema considerado [4]. Lo interesante es que el experimento con la selección aleatoria de centros es relativamente insensible

a la regularización; ese tipo de conclusión sugiere que el método ya es una forma implícita de regularización.

En el ajuste de la función **SinE** se usarán como centros los datos de entrada, con el objetivo de mostrar la capacidad de la reducción neuronales con la descomposición QLP. En el problema de clasificación de los tres tipos de dendidad de probabilidad se usarán los centros obtenidos por el parámetro $IW1,1$ de la función **newrb**, figura 1.1.

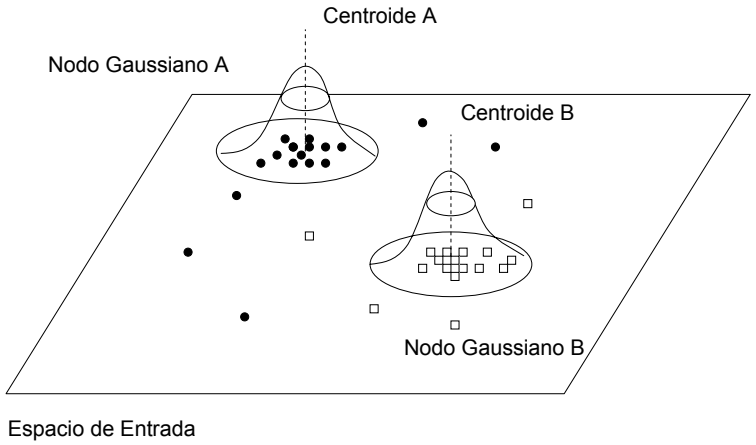


Figura 2.5: Respuesta localizada de dos neuronas ocultas

En la figura 2.5 se tiene la respuesta localizada de las neuronas ocultas en las RBF (nodos gaussianos). Los puntos representan patrones en el espacio de entradas, que en su mayoría se agrupan en torno a dos centros [58].

2.2.2. Una interpretación espacial de las RBF

Según se ha discutido hasta ahora, la capa oculta de una red RBF lleva a cabo una transformación no lineal sobre el espacio de las entradas. $\mathbf{x}$, de $\mathbb{R}^n \rightarrow \mathbb{R}^m$, donde m es el número de funciones de base radial (número de neuronas). Si las RBF, θ_j (la columna j de θ), tienen sus centros y anchuras fijados, entonces este mapeo es estático y el modelo global se puede interpretar como lineal en dichas funciones [68]. En esta configuración, los únicos parámetros de la red que se necesita definir son los coeficientes w_i.

Anteriormente fueron discutidas de manera superficial algunas características del modelo de esta forma. En esta sección, el problema de la definición del vector $\mathbf{w}$ se retoma con una interpretación un poco diferente. Para ello, se utilizan algunas nociones provenientes del álgebra lineal que permiten

una interpretación de naturaleza mas geométrica, en el caso particular que nos ocupa.

Recordamos que, dados n pares de entrenamiento, $(\mathbf{x}_i, d_i) : i = 1, \ldots, n$, se puede escribir el sistema de ecuaciones que se mostró en (1.1).

Nótese que para un espacio de salida multidimensional existirían varios sistemas lineales independientes, todos ellos compuestos por la misma matriz de diseño, pero con vectores, $\mathbf{d}$ y $\mathbf{w}$, específicos. Esta interdependencia de los sistemas lineales permite reducir toda la derivación considerando solamente el caso unidimensional. La extrapolación para el caso donde existe más de una neurona en la capa oculta es trivial.

Una forma interesante de interpretar el sistema lineal es considerar cada columna, θ_j, de la matriz de diseño como un vector perteneciente a $\mathbb{R}^p$. Cada uno de estos vectores correspondería a las activaciones de una RBF de la capa oculta de la red. Todas las combinaciones lineales posibles de las componentes de θ_j dan origen a un subespacio vectorial, $\theta \subseteq \mathbb{R}^p$ [36, 10][1]. La dimensión de ese espacio, θ, dependerá del número de vectores θ_j linealmente independientes. El vector $\mathbf{w}$ puede ser relacionado con θ_j de la siguiente manera:

$$\phi = w_1 \cdot \begin{bmatrix} \theta_{11} \\ \theta_{21} \\ \vdots \\ \theta_{p1} \end{bmatrix} + w_2 \cdot \begin{bmatrix} \theta_{12} \\ \theta_{22} \\ \vdots \\ \theta_{p2} \end{bmatrix} + \cdots + w_m \cdot \begin{bmatrix} \theta_{1m} \\ \theta_{2m} \\ \vdots \\ \theta_{pm} \end{bmatrix} .$$

La predicción de la salida, ϕ, del entrenamiento del conjunto de entrada, $x_i, i = 1, 2, \ldots, n$, por el modelo lineal usando la ecuación normal es:

$$\phi = \sum_{j=1}^{m} w_j \theta_j(x) = \theta_j^T w \tag{2.21}$$

donde θ es la matriz de diseño. Así, la salida de la red sería

$$\phi = \begin{pmatrix} \theta_1^T w \\ \theta_2^T w \\ \vdots \\ \theta_p^T w \end{pmatrix} = \theta w = \theta A^{-1} \theta^T d \tag{2.22}$$

donde

[1]Para prevenir una proliferación de notaciones, tanto el espacio vectorial generado por θ_j como la matriz de diseño que consta de estos vectores son referenciados como θ_j. La distinción se deriva fácilmente del contexto.

$$\widehat{w} = A^{-1}\theta^T d, \qquad A = \theta^T\theta \tag{2.23}$$

siendo $\widehat{w} = [\widehat{w}_1, \widehat{w}_2, \ldots, \widehat{w}_m]$ es el vector que minimiza la función de coste (ver Apéndice, A.1.4).

De esta forma, lo que se pretende es tomar el vector $\phi \in \mathbb{R}^p$ tan cercano como sea posible al vector $\mathbf{d} \in \mathbb{R}^p$. Llamando $\iota(\iota \leq p)$ al número de vectores θ_j (columnas) linealmente independientes que constituyen una base para el espacio vectorial θ [36], se pueden distinguir las siguientes situaciones:

- Si $\iota \geq p$, entonces todo el espacio vectorial, $\mathbb{R}^p$, se puede alcanzar, o sea, $\mathbb{R}^p \subseteq \theta$, y el sistema presenta por lo menos una solución exacta.
- Si $\iota < p$, entonces $\theta \subset \mathbb{R}^p$, esto es, no todos los vectores de $\mathbb{R}^p$ pueden ser representados a partir de la base formada por los ι vectores linealmente independientes.

En la práctica, lo que se observa en general es un número de puntos bastante mayor que el número de centros, o sea, $p \gg m \gg \iota$. Como ilustración para $p = 3$ y $m = 2$ se puede observar la figura 2.6. La consecuencia de este hecho es que el vector $\mathbf{d}$ no puede pertenecer al espacio θ, y por lo tanto la red puede ser incapaz de realizar la representación de manera exacta. En este caso, se debe adoptar un criterio que defina una medida de la distancia entre $\mathbf{d}$ y el vector ϕ computado por la red. Para ello, se define el vector de error, $\mathbf{e}$, de la forma:

$$\mathbf{e} = \mathbf{d} - \phi = d - \theta A^{-1}\theta^T d = (I_p - \theta A^{-1}\theta^T)d = Pd \ . \tag{2.24}$$

donde $P = I_p - \theta A^{-1}\theta^T$, es la matriz de proyección.

El cuadrado del error en el peso que minimiza la función de coste en términos de P y d es:

$$e^T e = \mathrm{SC} = (d-\theta)^T(d-\theta) = (Pd)^T(Pd) = d^T P^T P d = d^T P^2 d \tag{2.25}$$

Dicho vector, $\mathbf{e}$, representa la distancia euclídea entre la respuesta del vector deseado, (d), y la respuesta vectorial de la red, (ϕ), definida como:

$$\|\mathbf{e}\| = \|\mathbf{d} - \phi\| = \sqrt{\sum_{i=1}^{3}(d_i - \phi_i)^2} \ , \tag{2.26}$$

donde $\|\cdot\|$ representa la longitud o módulo del vector [36].

El problema pasa a ser, en esa interpretación, encontrar el vector ϕ (salida) que minimiza la longitud del vector error, $\mathbf{e}$. Se puede reformular este problema de forma equivalente como la minimización de $\|\mathbf{e}\|^2$, lo que correspondería a la función de coste ζ_{SSE}:

$$\|\mathbf{e}\|^2 = \mathbf{e}^T\mathbf{e} = \sum_{i=1}^{3}(d_i - \phi_i)^2 \ . \tag{2.27}$$

Se sabe que el vector $\phi \in \theta$ que presenta la menor distancia euclídea al vector $\mathbf{d} \in \mathbb{R}^p$ corresponde a la proyección ortogonal de este último en el espacio vectorial α. Tal proyección viene dada por la ecuación siguiente [36, 93]:

$$\mathbf{V} = \text{proj}_\alpha(\mathbf{d}) = \theta \cdot \theta^+ \cdot \mathbf{d} \ , \tag{2.28}$$

donde $\theta^+ = (\theta^T\theta)^{-1}\theta^T$ es la pseudoinversa de θ, como ya se ha discutido anteriormente.

La figura 2.6 ilustra en dos dimensiones lo expuesto para la situación donde $p = 3$. En ese caso, la red solamente podría alcanzar puntos pertenecientes a un plano fijo y nunca se ajustaría exactamente a $\mathbf{d}$ cuando el vector a aproximar estuviese fuera del plano. La mejor aproximación a $\mathbf{d}$ —aquella que minimiza $\mathbf{e}^T\mathbf{e}$— correspondería a la proyección ortogonal de $\mathbf{d}$ sobre el plano α. La diferencia, $e = \hat{d} - v$, debe ser ortogonal a v, y por tanto a θ, esto es, $\theta'(\hat{d} - v) = 0$. Por tanto, si $d = v$, $e = 0$. Es decir, la proyección del vector $\hat{d}$ sobre α se obtiene multiplicando el vector d por la matriz $\theta(\theta'\theta)^{-1}\theta'$.

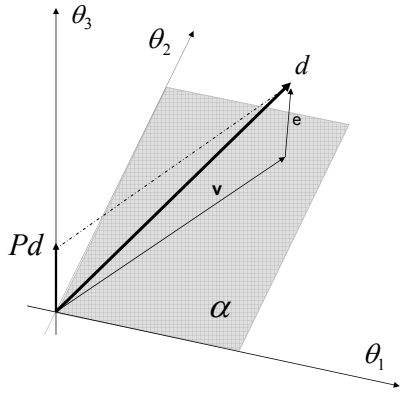


Figura 2.6: Proyección ortogonal del vector $\mathbf{d}$ en el subespacio vectorial θ.

Una pregunta que aparece inmediatamente es: ¿cómo determinar $\mathbf{w}$ de modo que $\mathbf{y}$ represente la proyección ortogonal de $\mathbf{d}$ sobre ϕ? Una manera de

ver la ecuación (2.28) es considerar cada columna, θ_j, de θ como ponderada por un elemento del vector $\theta^+ \cdot \mathbf{d}$. No es difícil percibir que la solución, $\mathbf{w}$, para el problema lineal viene dada por $w = (\theta)^+ \cdot \mathbf{d}$, habida cuenta de que $w = \left(\theta^T\theta\right)^{-1}\theta^T d = A^{-1}\theta^T d$.

Esta solución es la misma encontrada para el caso en que se usa el SSE como función de coste, según lo discutido. Tal conclusión no sorprende, dada la equivalencia entre esa medida y la medida de la distancia euclídea, como se ha resaltado anteriormente. Lo importante, en este contexto, es la nueva interpretación dada al problema: en vez de verlo como un problema del ajuste en que se pretende minimizar una función de coste, puede ser entendido como la aproximación de un vector, $\mathbf{d} \in R^p$, a partir del vector $\mathbf{y}$, perteneciente a un espacio vectorial, θ, de dimensión mas baja. Esa interpretación espacial del problema facilita la introducción de los conceptos fundamentales para el entendimiento del algoritmo de mínimos cuadrados ortogonales (OLS, *Orthogonal least squares*) [2].

2.3. Descomposiciones QR y QLP

La matriz A de tamaño $m \times n$ se puede descomponer como $A = QR$, factorización que fue introducida por [5] precisamente en el contexto de problemas de ajuste de modelos lineales por mínimos cuadrados [41]. La matriz, Q, obtenida es ortogonal y R es una matriz triangular superior, $R(i,j) = 0, i > j$. Para que el producto QR sea posible, Q será $m \times n$. En el caso en que rango$(A) < n$ (A es de rango deficiente), tenemos el problema de mínimos cuadrados de rango no completo.

Para tratar el caso del rango deficiente de A fue propuesta por [5] la descomposición QR con pivoteo. La versión descrita en esta referencia estaba basada en el uso de transformaciones de Householder. La diferencia con la QR original reside en el uso de una matriz de permutación, P, de forma que el efecto final equivale a un tipo especial de descomposición QR sobre una versión de A con algunas columnas intercambiadas, $QR = AP$.

Sea $\mathbf{A}$ una matriz $n \times p$ con $n \geq p$. Entonces, para cualquier matriz de permutación, Π_R, hay una matriz ortogonal, $\mathbf{Q}$, tal que:

$$\mathbf{Q}^T \cdot \mathbf{A} \cdot \Pi_R = \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix} \quad . \tag{2.29}$$

donde $\mathbf{R}$ es una matriz triangular superior. La matriz Π puede elegirse de tal modo que los elementos de $\mathbf{R}$ cumplan

$$r_{kk}^2 \geq \sum_{i=k}^{p} r_{ij}^2 \qquad j = k+1, \ldots, p \tag{2.30}$$

En otras palabras, si R_{kk} denota la submatriz de huella (*trailing*) de $\mathbf{R}$ de orden $p-k+1$, entonces, la norma de la primera columna de R_{kk} domina las normas de las otras columnas. Esta descomposición se denomina descomposición QR con pivoteo.

Específicamente, en el paso k de la reducción, se dispone de las $k-1$ transformaciones de Householder, $H_1, H_2, \ldots, H_{k-1}$, y sus permutaciones correspondientes, $\Pi_1, \Pi_2, \ldots, \Pi_{k-1}$, de tal modo que

$$H_{k-1} \cdots H_1 \Pi_1 \cdots \Pi_{k-1} = \begin{pmatrix} R_{11} & R_{12} \\ 0 & A_k \end{pmatrix} \tag{2.31}$$

donde R_{11} es una matriz triangular superior.

Descomposición QLP con pivoteo

Consideremos el factor de la descomposición QLP con pivoteo, R, particionado de la siguiente forma:

$$R = \begin{pmatrix} r_{11} & r_{12}^T \\ 0 & R_{22} \end{pmatrix} \tag{2.32}$$

Ya sabemos que r_{11} es una subestimación de la norma euclídea de la matriz $\mathbf{A}$. Una estimación mejor es la norma, $\ell_{11} = \sqrt{r_{11}^2 + r_{12}^T r_{12}}$, de la primera fila de $\mathbf{R}$. Se puede calcular dicha norma multiplicando por la derecha por una transformación de Householder, $\mathbf{H}_1$, que reduce la primera fila de $\mathbf{R}$ a un múltiplo de $\mathbf{e}_1$, siendo $\mathbf{e}_1$ un vector cuyas componentes son todas uno.

$$\mathbf{R} \cdot \mathbf{H}_1 = \begin{pmatrix} \ell_{11} & \mathbf{0} \\ \boldsymbol{\ell_{12}} & \hat{\mathbf{R}}_{\mathbf{22}} \end{pmatrix} \ . \tag{2.33}$$

Se puede obtener un valor mejor si intercambiamos la fila de mayor longitud (euclídea) de $\mathbf{R}$ con la primera fila:

$$\boldsymbol{\Pi}_1 \cdot \mathbf{R} \cdot \mathbf{H}_1 = \begin{pmatrix} \ell_{11} & \mathbf{0} \\ \boldsymbol{\ell_{12}} & \hat{\mathbf{R}}_{\mathbf{22}} \end{pmatrix} \ . \tag{2.34}$$

Si ahora trasponemos esta ecuación, se observa que es el primer paso de la triangulación de Householder con pivote aplicado a $\mathbf{R}^T$.

Si continuamos con esta reducción y transponemos el resultado, se obtiene una descomposición triangular de la seguiente forma:

$$\mathbf{\Pi}_L^T \mathbf{Q}^T \mathbf{A} \mathbf{\Pi}_R \mathbf{P} = \begin{pmatrix} \mathbf{L} \\ \mathbf{0} \end{pmatrix} \quad . \tag{2.35}$$

Esta es la denominada descomposición QLP con pivoteo de $\mathbf{A}$ y a los elementos de la diagonal de $\mathbf{L}$ los llamaremos *L-valores* de $\mathbf{A}$.

Esta metodología de la descomposición QLP con pivoteo sugiere que previsiblemente suministrará mejores aproximaciones de los valores singulares de la matriz A que los de la descomposición QR con pivoteo. En el experimento se mostrará que esta descomposición indica el valor singular con notable fidelidad.

Algoritmo de descomposición QLP Para nuestros propósitos, dada una matriz, $\mathbf{A}$, de dimensión $n \times p$, el algoritmo para calcular la descomposición QLP se puede implementar mediante dos pasos o llamadas al procedimiento QR (supuesto que esté ya disponible) tal y como sigue [86]:

1. Calcular la descomposición QR de la matriz original, $\mathbf{A}$, obteniendo con ello los factores, $\mathbf{Q}$ y $\mathbf{R}$, habituales, así como una permutación que denominaremos $\mathbf{Q}_R$.

2. Desechar el factor $\mathbf{Q}$, pues no es necesario para el cálculo de la QLP[2].

3. Realizar una nueva descomposición QR sobre el factor, $\mathbf{R}$, obtenido en el primer paso, obteniendo un nuevo factor del tipo $\mathbf{R}$ (al que denotaremos $\mathbf{L}$) y una nueva permutación, $\mathbf{Q}_L$, que no nos interesa.

4. La permutación $\mathbf{Q}_R$ indica la significación relativa de las columnas en $\mathbf{A}$ (exactamente igual que sucedía con la rutina QR). Sin embargo, el factor $\mathbf{L}$ obtenido en el paso tercero indica de manera más robusta y fiable el *gap* o salto numérico que permite llevar a cabo la detección del número concreto de componentes significativas que existen en $\mathbf{A}$. Es decir, obtenemos una información más precisa (en general) del rango numérico de $\mathbf{A}$ usando QLP que usando QR, y tan precisa como observando la diagonal obtenida con SVD (pero con la ventaja, en el caso de QLP, de sus menores requerimientos computacionales). Más adelante describiremos un interesante experimento sintético que ilustra las diferencias mencionadas (Sección 4.1.1).

[2]Por supuesto, se entiende que en una implementación real en computador, se debe intentar recurrir (en la medida de lo posible) a rutinas que eviten el cálculo explícito del factor, $\mathbf{Q}$, o a modificar las rutinas ya disponibles eliminando el cómputo de dicho factor.

Los elementos de la diagonal, R, se llaman R-valores de la matriz A; y las diagonales de L se llaman L-valores de dicha matriz. En este ejemplo observaremos que se obtiene el valor singular de A con considerable fidelidad y que la descomposición QLP es dos veces más rápida que la descomposición QR.

QLP para selección de entradas La descomposición QLP resulta apropiada para la selección de regresores al ajustar modelos de series caóticas a datos disponibles 4.5. Hay una sutil pero importante diferencia entre la aplicación del SVD, QR y QLP para los siguientes propósitos:

1. La información sobre una base adecuada para el subespacio vectorial de dimensión $r, r < n$, generado por las columnas de una matriz, $A_{m\times n}$, de rango deficiente. Aquí se escogen las columnas seleccionadas según el orden del QLP con las misma dimensión $m \times n$.

2. La selección de columnas de A. Aquí el enfoque es distinto, pues el objetivo principal es obtener un subconjunto de columnas de la propia matriz A, de tal modo que la selección incluya, por decirlo de algún modo, las r columnas *mejores* de A, usando QLP, respecto del criterio de independencia lineal.

Así, se puede demostrar que el QLP es una herramienta adecuada para la selección de columnas significativas en una matriz general A. Usando la descomposición QR, la demostración puede verse en [30]. El QLP es una extensión del QR como fue detallado anteriormente y se utilizará en el contexto del algoritmo de clasificación y predicción que estamos presentando.

Si nos limitamos a realizar una descomposición SVD sobre la matriz $A(t_o)$, obtendremos una versión filtrada de la misma, en el sentido de que se descartará la información ligada a los $W - N_k$ valores menos significativos del factor S obtenido con SVD. Estaremos obteniendo una solución proyectada en un espacio de datos de dimensión $N_k, N_k < W$, y los datos proyectados tenderán a cancelar el posible efecto de ruido numérico en las medidas de la serie caótica considerada. En [30] se menciona que el rango deficiente en $A(t_o)$ en general puede deberse también a la redundancia en los datos, en lugar de presentarse únicamente a partir de consideraciones numéricas [81].

El teorema siguiente, de [30] y reformulado por [81], proporciona la ligadura fundamental entre SVD y QLP para la resolución del problema de selección de columnas de A.

Teorema 2.3.1 *Sea $A(t_o) = USV'$ la descomposición SVD de la matriz $A(t_o)$ y sea P una matriz arbitraria de permutación de orden W, donde W*

es el tamaño de la ventana en el predictor neuronal P. Supongamos que la matriz $A(T_o)$, tras la aplicación de P, se expresa en la forma $A(t_o)P = [C_1C_2]$ donde C_1, tiene $r = N_p$ columnas y C_2 tiene $W - r$ columnas; suponiendo que la submatriz principal de orden r de P^TV, denominada $\bar{F}$, es invertible, los valores singulares de $A(t_o)$ y de C_1 verifican

$$\frac{\sigma_{p(r)}(A(t_o))}{\| \bar{F}^{-1} \|} \leq \sigma_{p'(r)}(C_1) \leq \sigma_{p(r)}(A(t_o)) \tag{2.36}$$

donde $\sigma_{p(r)}(\cdot)$ denota el r-ésimo mayor valor singular de la matriz que aparece entre paréntesis, cuando los valores singulares se ordenan decrecientemente según la permutación p (para $A(t_o)$); para la matriz C_1 se supone una permutación distinta p'.

La base de la descomposición QLP es la descomposición QR. Se sabe que $A = Q.R.pqr'$. Si hacemos la factorización QR de R' se tiene $R = Q1.R1.pqr1'$. Usando estas dos descomposiciones se obtiene la descomposición QLP, cuya diagonal, $R1$ se aproxima más a la diagonal S de la descomposición SVD que la diagonal R de la descomposición QR. Así, tenemos la factorización del QLP de la siguiente forma:

$$Aqlp = Q.R1'.Q1 = A \tag{2.37}$$

Escribiendo la descomposición QLP en forma compacta, se obtienen las matrices ortogonales $P1$ y $Q2$, donde $L2$ es una matriz diagonal, y pr es el orden de las columnas más relevantes es esta descomposición. La inversa de la matriz A usando las componentes QLP sería:

$$A = P1.(L2^{-1}).Q2 \tag{2.38}$$

Nos ocupamos ahora de descomposiciones y factorizaciones matriciales que se presentan en el caso sobredimensionado (matrices $m \times n$, donde $m > n$). La descomposición QLP puede emplearse tanto para propósitos de análisis y reducción de dimensionalidad de los datos, como para determinación de subconjuntos de entradas y de elementos de proceso que contengan la mayor parte de la información relevante en el contexto de aplicaciones de predicción de serie caóticas con RBF reducido por QLP, como se sugiere en el capítulo 4.1.1. La motivación para el uso de la factorización QLP en algebra lineal numérica reside en ciertos problemas que se plantean en regresión lineal, concretamente los llamados problemas cercanos a la singularidad. Afortunadamente, la mayoría de los programas de ordenador disponibles para la solución de problemas que involucran sistemas lineales sobredeterminados tiene en cuenta el caso mencionado; el problema por tanto reside en cómo

operar cuando la matriz A es invertible teóricamente, pero cercana a la singularidad.

Pseudocódigo del algoritmo rbfQLP Considerando la figura 1.1 y el pseudocódigo RBFqlp, presentamos una descripción algorítmica (en pseudocódigo básico) del nuevo procedimiento para clasificación dos a dos de las densidades de probabilidad Weibull, Lognormal y Gamma.

1. Construir la matriz A usando las dos densidades consideradas
2. Construir la matriz deseada, T, correspondiente a las dos densidades While (true)
3. Determinar una matriz de tamaño 1000×8, donde 1000 corresponde a las dos densidades de tamaño 500, y 8 a las características descriptivas (ver tabla 4.6)
4. Tomar el vector de entrada y calcular la salida de la red de acuerdo con 2.14 $\rightarrow$ Calcular una clasificación $\Phi(x) \rightarrow$ densidad 1 o densidad 0
5. Tomar nuevas entradas, P, reducidas según los índices, L, de la descomposición QLP
6. Reentrenar la red con entradas reducidas (características descriptivas) según el QLP y tambien no reducidas
7. Usar QLP para indicar el número de neuronas RBF más relevantes
8. Eliminar toda RBF que no haya sido marcada en la reducción (*pruning*)
9. Reentrenar la red con las neuronas que han sobrevivido
10. Calcular la salida de la red, $\phi(x)$, de acuerdo con la ecuación 2.11
11. Calcular el error de acuerdo con la ecuación 2.27
12. Comparar el resultado de la red reducida con la no reducida a través del índice de error aparente, APER, y el índice de clasificación correcta, CCR (ver tabla 3.6)

Pseudocódigo del algoritmo LagQLP Considerando la figura 1.1 y el pseudocódigo LagQLP, presentamos una descripción algorítmica (en pseudocódigo basico) del nuevo procedimiento de ajuste de la serie caótica, donde v denota la iteración actual y k el horizonte de predicción.

1. Iniciar las variables.
2. Formar la matriz A usando la ventana de retardos
3. Encontrar las entradas $[v_1, v_2, v_3, \ldots, v_k]$ a través del QLP
4. Determinar los indices de retardos, $L = [L_1, L_2, \ldots, L_{Nk}]$, donde Nk es la última columna de la ventana de retardos
5. Entrenar la red con entradas reducidas y también con entradas no reducidas, usando la ecuación 2.11
6. Calcular la salida de la red, $\phi(x)$, de acuerdo con la ecuación 2.11
7. Calcular el error de acuerdo con la ecuación 2.27
8. Comparar el resultado de la red reducida con la no reducida.

Pseudocódigo del Algoritmo rbfHQLP Considerando la figura 1.2 y el pseudocódigo SinEQLP, presentamos una descripción algorítmica (en pseudocódigo basico) del nuevo procedimiento del ajuste de la función específica SinE.

1. Generar 100 valores de entrenamiento según $y = z(x)+\epsilon$, y 1000 valores para comprobación
2. Formar la matriz H (centros y radios) dependiendo de la naturaleza de la RBF (ver ecuaciones 2.13, 2.14 y 2.15)
3. Usar QLP para encontrar las ramas de recorte (*size-prune*) más relevantes, o sea, encontrar los índices de las columnas (neuronas) más significativas según el QLP, a partir de la gráfica de la diagonal L del QLP.
4. Formar la matriz H reducida usando los valores adecuados según el índices de columnas (neuronas) más relevantes
5. Entrenar la red con todas las neuronas RBF de acuerdo con la naturaleza, y posteriormente entrenar la red según la reducción propuesta por el QLP.
6. Eliminar toda RBF que no haya sido marcada en la reducción
7. Reentrenar la red con las neuronas que hayan sobrevivido
8. Calcular la salida de la red, $\phi(x)$, de acuerdo con la ecuación 2.11
9. Calcular el error de acuerdo con la ecuación 2.27

10. Comparar el resultado usando el error de predicción final, y el criterio de información de Schwarz, BIC (ver tabla 3.6)

11. Comparar el resultado de la red reducida con la no reducida.

2.4. Las distribuciones Gamma, Lognormal y Weibull

2.4.1. Distribución Gamma

Una variable aleatoria, X, tiene una distribución Gamma(α, λ) cuando su función de densidad es:

$$f(t) = \frac{\lambda^{\alpha} t^{\alpha-1}}{\Gamma(\alpha)} dt \tag{2.39}$$

donde $\alpha, \alpha > 0$, es el parámetro de forma y $\lambda, \lambda > 0$, es el parámetro escalar. La función $\Gamma(\alpha)$ se llama función gamma completa y está definida como:

$$\Gamma(\alpha) = \int_0^{+\infty} e^{-t} t^{\alpha-1} dt, \alpha > 0 \tag{2.40}$$

La función de distribución de probabilidad, $F(t)$, de esta gamma está dada por:

$$T(t) = \frac{1}{\Gamma(\alpha)} \int_0^{\lambda t} e^{-u} u^{\alpha-1} du, t \geq 0 \tag{2.41}$$

Es importante que se presente a la RBF distribuciones con los mismos coeficiente de media y coeficiente de variación. Puesto que la cantidad es dimensional, es una medida útil para la variabilidad de la variable aleatoria X [77] . Aquí, se utilizará el cuadrado del coeficiente de variación, C_x^2, más que C_x.

La media y el cuadrado del coeficiente de variación de una gamma son

$$E(X) = \frac{\alpha}{\lambda} \tag{2.42}$$

$$C_X^2 = \frac{1}{\alpha} \tag{2.43}$$

Este resultado muestra que la gamma puede ser ajustada para cada variable aleatoria positiva en los dos primeros momentos. Antes de presentar

a la RBF reducida es importante verificar los dos parámetros. La gamma es siempre unimodal. Para el caso $C_x^2 < 1$ la densidad crece hasta un máximo en $t = \dfrac{\alpha - 1}{\lambda} > 0$ y después decrece hasta cero cuando $t \to \infty$; mientras que para el caso $C_x^2 \geq 1$ la densidad tiene el máximo en $t = 0$ y luego decrece de $t = 0$ en adelante.

2.4.2. Distribución Lognormal

Una variable aleatoria, X, se dice lognormal si tiene la siguiente densidad de probabilidad:

$$f(t) = \frac{1}{\alpha t\sqrt{2\pi}} \exp\left[-\frac{1}{2\alpha^2}\left(\ln(t) - \lambda\right)^2\right], t > 0 \tag{2.44}$$

donde el parámetro de forma, α, es positivo y el parámetro λ puede tomar cualquier valor real. La función de distribución es

$$F(t) = \theta\left(\frac{\ln(t) - \lambda}{\alpha}\right), t > 0,$$

donde

$$\theta(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{\mu^2}{2}\right) \mathrm{d}\,\mu \tag{2.45}$$

La media y el cuadrado del coeficiente de variación son:

$$E(X) = \exp\left(\lambda + \frac{\alpha^2}{2}\right) \tag{2.46}$$

$$C_x^2 = \exp\left(\alpha^2\right) - 1 \tag{2.47}$$

Por tanto existe una única distribución lognormal dados los dos primeros momentos. La densidad lognormal es unimodal con un máximo en $t = \exp(\lambda - \alpha^2)$.

2.4.3. Distribución Weibull

Una variable aleatoria, X, tiene una distribución Weibull cuando tiene la densidad de probabilidad:

$$f(t) = \alpha\lambda(\lambda t)^{\alpha-1} \exp[-(\lambda t)^{\alpha}], t > 0 \tag{2.48}$$

donde $\alpha, \alpha > 0$ es el parámetro de forma y $\lambda, \lambda > 0$ es el parámetro escalar. La correspondiente función de distribución de probabilidad viene dada por $F(t) = 1 - \exp[-(\lambda t)^{\alpha}], t \geq 0$.

La media y el cuadrado del coeficiente de variación son:

$$E(X) = \frac{1}{\lambda}\Gamma\left(1 + \frac{1}{\alpha}\right) \tag{2.49}$$

$$C_x^2 = \frac{\Gamma(1 + 1/\alpha)}{[\Gamma(1 + 1/\alpha)]^2} - 1 \tag{2.50}$$

La densidad Weibull es siempre unimodal con un máximo en $t = \lambda^{-1}(1 - \frac{1}{\alpha})^{\frac{1}{\alpha}}$ si $C_x^2 < 1$, y en $t = 0$ si $C_x^2 \geq 1$ ($\alpha < 1$).

La figura 2.7 ilustra estos hechos considerando las densidades gamma, lognormal y Weibull para un coeficiente de variación $C_x^2 = 0{,}25$, y media $E(X) = 1$.

En los experimentos se verá que se requiere una gran cantidad de neuronas RBF cuando se intenta diferenciar estas densidades.

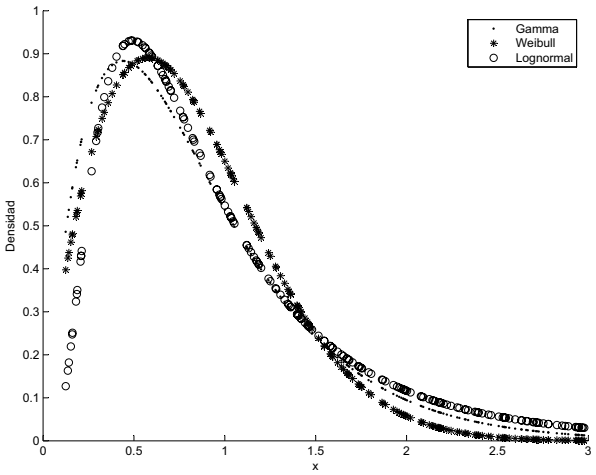


Figura 2.7: Densidades gamma, lognormal y Weibull

Las densidades Gamma y Weibull son muy similares en forma, y para $C_x^2 < 1$ la densidad Lognormal adopta también una forma muy similar. Las diferencias entre las tres densidades son más significativas en el comportamiento de las colas de dichas distribuciones.

La descomposición QLP para reducción de neuronas es un método eficiente para diferenciar estas tres densidades usando las características descriptivas de las mismas. [16].

2.5. Mapa de Hénon

Las RBF son aplicables a una gran variedad de problemas. [4] y [45] usaron RBF sucesivamente para la predicción de una serie caótica. Aquí, se usará la serie caótica llamada Mapa de Hénon.

Consideremos un sistema caótico no lineal sin ruido como el mapa de Hénon [37], definido por:

$$X_{n+1} = 1 - 1{,}4X_n^2 + 0{,}3X_{n-1} \tag{2.51}$$

$$Y_{n+1} = X_n \tag{2.52}$$

Capítulo 3

Algoritmos de reducción

3.1. Introducción

Recordamos que el objetivo principal de nuestra tesis es contribuir al diseño de redes neuronales artificiales óptimas de tipo RBF. La optimalidad se entenderá en el sentido de una adecuación de los recursos neuronales (selección del número de neuronas) de forma automática, acorde con el contenido informativo de los datos y de los resultados intermedios de procesamiento. Las herramientas que usaremos para fundamentar nuestros algoritmos serán descomposiciones espectrales de las matrices de datos involucradas, dando siempre especial importancia al estudio de la complejidad computacional en que se incurre, buscando su minimización en la medida de lo posible.

Nuestros desarrollos se diferencian de los algoritmos básicos para redes RBF ya existentes en la literatura, en varios aspectos:

1. **Reducción neuronal.** En primer lugar, como acabamos de mencionar, en la mayoría de artículos propuestos hasta la fecha se lleva a cabo una primera fase de posicionamiento inicial de los parámetros de las neuronas en la red RBF mediante técnicas clásicas tales como el *clustering* consiguiéndose redes por lo general sobredimensionadas a la hora de la generalización neuronal.

   Por el contrario, en nuestros algoritmos preferimos añadir **técnicas de reducción de las estructuras neuronales** que manejen la complejidad resultante, simplificando entradas y neuronas redundantes o poco significativas a través del QLP, y la adecúen más al verdadero contenido informativo de los datos y de los procesos neuronales. De esta forma nos aprovechamos de la simplicidad de las técnicas de *clustering*[1], pero

[1]Un conjunto de neuronas con salidas binarias, de las cuales solo una está activa en

consiguiendo un control más preciso sobre la complejidad y parsimonia de los modelos neuronales resultantes.

2. **Implementabilidad en tiempo real.** También prestamos una atención especial al aspecto de **implementación de los modelos de redes RBF en contextos de operación en tiempo real**, considerándolos siempre como susceptibles de integración en un sistema operativo mayor de toma de decisiones y predicción. Creemos que es importante, a la hora de idear los algoritmos neuronales, tener en mente la posible implementación en el mundo real, puesto que ello determinará la elección y el énfasis en unas u otras alternativas de aprendizaje, entrenamiento y comprobación aplicables a cualquier problemática del mundo real en que este tipo de aproximación tenga sentido.

   Por ejemplo, operando en tiempo real es impensable proceder a un reentrenamiento completo de una nueva red cuando, por ejemplo, cambian las características estadísticas de los datos de entrada de forma que se requiera un cambio estructural (topológico) de la misma. Sería conveniente que la red detectase de alguna manera en tiempo real dichos cambios, y, por eso, a la hora de diseñar rutinas neuronales donde queremos efectuar cambios repentinos en la topología, conviene más idear métodos capaces de modificar la propia red que se está empleando, y de la forma más eficiente posible, que seguir el camino fácil de desechar dicha red, y proceder a reentrenar una nueva desde cero.

3. **Complejidad computacional.** La consideración de implementabilidad de las rutinas neuronales también implica un esfuerzo mayor de estudio en el sentido de **exigencia computacional de los modelos**. Nuestros métodos, como se verá, obtienen sus ventajas de ciertas manipulaciones matriciales que pueden suponer en principio una desventaja (computacionalmente hablando) con respecto a otros algoritmos establecidos en la literatura.

   Es más, lejos de contentarnos con nivelar la balanza, de hecho nuestro esfuerzo también se ha dirigido a estudiar la manera de reducir esta carga computacional, incidiendo lo menos posible en el rendimiento funcional de la red neuronal. La discusión muestra que existen incluso varias formas alternativas de implementar la reducción neuronal, pudiendo elegirse una u otra dependiendo de los requisitos de operación una vez que la red neuronal estuviese operando en un contexto o sistema más amplio, con restricciones de cómputo, tiempo, etc.

cada instante, nos puede decir a qué categoría pertenece la entrada actual

3.2. Consideraciones técnicas para el entorno Matlab®

Una vez detalladas las dificultades presentes en la modelización neuronal, es necesario profundizar un poco más en su desarrollo e implementación, desde la óptica de la simulación [43]. En los programas que acompañan a este trabajo, se ha pretendido utilizar diversos métodos matemáticos matriciales que soportan una reducción eficiente de los modelos neuronales, resolviendo de este modo gran parte del problema asociado con la determinación del número de entradas y/o del número de neuronas en los modelos de redes RBF. Una ventaja adicional de los métodos que proponemos es que poseen una firme base matemática, derivada y demostrada para nuestros propósitos a partir de estudios más generales de computación numérica matricial. La idea principal es disponer los sucesivos datos operativos de entrada e intermedios de la red neuronal en una matriz numérica, de forma que podamos aprovechar las consideraciones sobre el rango numérico de matrices a la hora de determinar la relevancia relativa de las diversas entradas al modelo, y de las propias neuronas existentes en la capa oculta de la RBF.

En primer lugar, por tanto, nos detenemos en idear un procedimiento con base teórica que permita la identificación correcta de las entradas y/o neuronas más importantes a través de factorizaciones matriciales mediante QLP.

En segundo lugar, llevar a cabo una implementación eficiente en ordenador de dicho procedimiento, empleando los programas ya disponibles en el entorno de computación matricial Matlab® , y añadiendo nuestros propios programas para aquellas funciones o cómputos de los que no exista aún equivalente directo implementado en dicho entorno.

En tercer lugar, nos detendremos a considerar la consecuente modificación de la estructura interna de las redes, las cuales se encontrarán incorporadas en el entorno de computación matricial Matlab® , accediendo a las estructuras de datos internas a las variables de tipo `net` en dicho entorno, reduciendo de forma efectiva las propias redes de forma que sea factible una implementación hipotética en un contexto de operación en tiempo real de todo el sistema.

La eliminación de entradas innecesarias de la red supone que los accesos a las conexiones establecidas con dichas entradas son superfluos, inhabilitándolos y haciendo depender a todas las neuronas del nuevo subconjunto de entradas que se haya determinado mediante la aplicación de las descomposiciones matriciales.

De la misma forma, la eliminación de neuronas redundantes se realizará accediendo a las estructuras neuronales en lenguaje Matlab® , elimi-

nando las conexiones que ya no tengan sentido al desaparecer las neuronas asociadas a ellas, y siendo necesario tan sólo reentrenar la neurona final de salida (sus coeficientes lineales, $w2$) lo cual se puede realizar de forma eficiente con una simple inversión matricial, que numéricamente además se ve facilitada por la propia eliminación del material neuronal redundante que de lo contrario daría lugar, en general, a dependencias lineales entre las salidas de las diversas neuronas.

3.3. Determinación del número de capas de la red

Resultan importantes distintas consideraciones sobre la estructura general de un modelo neuronal, previamente a considerar su implementación o entrenamiento:

- La primera de ellas consiste en que, de forma habitual, se definen modelos con al menos una capa oculta, debido a la limitación que posee un modelo neuronal con solo dos capas, entradas y salidas. El único modelo neuronal que hemos visto sin capa oculta propiamente dicha, es el modelo Adalina o Madalina, cuyas limitaciones vimos anteriormente en esta memoria (de hecho, ya vimos que se trataba de modelos isomorfos a un modelo simple de regresión lineal).

- En segundo lugar, se ha demostrado que los modelos neuronales con un máximo de dos capas ocultas pueden aproximar un conjunto particular de funciones con una exactitud arbitraria y que con una sola capa oculta es suficiente para aproximar cualquier función continua [40, 97].

- En tercer lugar, la elección del número de capas ocultas representa un compromiso de forma que, si es demasiado pequeño, el modelo obtenido puede no aproximar con la exactitud deseada, pero si es demasiado grande, se puede producir un sobreajuste, *overfitting*, que puede evitar el proceso de generalización en la fase de comprobación, es decir, fuera de la muestra utilizada para el aprendizaje, generando un modelo sobreparametrizado.

3.4. Determinación del número de neuronas ocultas en la red

¿Cómo afecta el número de neuronas ocultas de la red a la resolución de los problemas de ajuste y clasificación? Es decir, ¿puede conseguirse el mismo nivel de aproximación disminuyendo el número de neuronas ocultas? Para contestar a estas preguntas hemos realizado nuevas simulaciones, utilizando diferentes números de neuronas ocultas. En las tablas del experimento 4.4, se observa que, algunos números de neuronas ocultas no son suficientes para aproximar la función **SinE**, mientras que utilizar otros números de neuronas no afecta a los resultados. En el caso del ajuste usando la RBF tipo Gaussiana (veáse 4.4) se puede decir que la red con 51 neuronas ocultas es adecuada para aproximar la función **SinE**.

En este trabajo se muestra una red con 3 capas (entrada, oculta y salida) que está completamente conectada, ya que todas las neuronas de cada una de las capas están conectadas con todas las neuronas de la capa siguiente.

La estruturación en capas incrementa notablemente el poder representativo de las RNA (o capacidad de la red para modelar una función específica, en nuestro caso **SinE**). Esta afirmación se basa en el teorema de Aproximación Universal [40] y [32], que establece que una sola capa intermedia es suficiente para aproximar, con una precisión arbitraria, cualquier función con un número finito de discontinuidades, siempre y cuando las funciones de activación de las neuronas ocultas sean no lineales. El teorema establece que las redes multicapa no añaden capacidad a menos que la función de activación de las capas sea no lineal. La demostración intuitiva es sencilla. Para ello, se denota por W a la matriz de pesos de las interconexiones entre la capa de entrada y la primera capa intermedia, donde la fila k-ésima de dicha matriz se corresponde con el vector de pesos, w_k, asociado a la k-ésima neurona oculta. De esta forma, si X es el vector de entrada a la red y no hay funciones de activaciones no lineales, la salida de la primera capa intermedia vendrá dada por el producto W_1X. La salida de la segunda capa será $W_2(W_1X)$, y así sucesivamente. Como el producto de matrices es asociativo, la expresión anterior es equivalente a $(W_1W_2)X$, y se puede concluir, por tanto, que una red bicapa sería equivalente a una red monocapa con una matriz de pesos igual a la matriz resultado de $(W_1W_2$. La extensión a n capas es trivial.

3.5. Procedimientos de diseño de redes óptimas

La diversidad de métodos propuestos hasta el momento para abordar el diseño de redes óptimas, se puede sintetizar en tres grandes grupos [75]:

1. El primer grupo de técnicas descansa sobre la idea de que el número de nodos ocultos, o la complejidad general de un modelo neuronal, debe hacerse depender de alguna forma del **tamaño de la muestra utilizada en el proceso de estimación**, siendo establecida esta relación *a priori*, como por ejemplo: el número de conexiones debería ser inferior a un 10 % del tamaño de la muestra, n, o el número de unidades ocultas debe ser del orden de $O(n)$ o $O(\log n)$. El problema principal de estas técnicas es que realizan un **análisis claramente estático y precisan de un análisis previo** de la dimensionalidad del vector de entradas y de la cantidad de datos disponibles. En aplicaciones reales esto puede ser algo indeterminado. Debido a esta limitación sólo pueden proporcionar una estimación muy aproximada del tamaño de la capa oculta o del número óptimo de elementos neuronales. Estos métodos, por otro lado, **no contemplan la posibilidad de adecuación de la red neuronal a cambios en la dinámica** o en la caracterización estadística del proceso subyacente a los datos de entrada.

2. El segundo grupo abarca una serie de **técnicas constructivas**, tales como la correlación en cascada [22], algoritmos de *tiling* [60], árboles de decisión neuronal [27], algoritmos *upstart* [26], o el procedimiento CLS [74]. Estos métodos constructivos realizan de forma secuencial su proceso, introduciendo una a una las diferentes capas y neuronas a medida que el modelo las necesita. Tal y como comenta [75], estas técnicas garantizan la convergencia del modelo hacia su generalización pero no su estabilidad (es decir, el número de capas o elementos se puede disparar indefinidamente, conduciendo al *overfitting*, si no se controla adecuadamente el proceso).

3. Por último, las técnicas que suponen una **reducción paulatina de los modelos**, operan lógicamente en la dirección opuesta (si bien puede conjugarse su operación con las técnicas de crecimiento o constructivas). Su tarea es ir reduciendo la red y eliminando las conexiones redundantes o con menor sensibilidad (la definición concreta de esa sensibilidad depende del algoritmo concreto que se considere). Este grupo incluye las siguientes técnicas: reducción de modelo en dos etapas [84], selección artificial [39], y sensibilidad basada en el error. La

idea común es la exclusión paulatina de pesos del modelo o incluso de unidades neuronales completas [46], aunque no siempre es posible una reducción óptima.

3.6. Criterios de evaluación

El tema de Redes Neuronales necesita un tratamiento formal, con el objetivo de generalizar su tratamiento y delimitar su alcance, así las redes necesitan de un criterio de evaluación que permita comparar el funcionamiento de modelos alternativos y la selección del mejor.

Es importante observar que el rendimiento para las muestras de comprobación tras el aprendizaje de cualquier modelo neuronal, debe ser adecuadamente evaluado mediante al menos alguna de las medidas reseñadas en la tabla siguiente, y, en general, comprobar (si es necesario) si dicho rendimiento es o no mejor que el de sus homólogas en modelos estadísticos tradicionales.

Criterio	Expresión
Error cuadrático medio, MSE	$\frac{1}{N}\sum_{i=1}^{N}(d_i - \hat{d}_i)^2$
Raíz cuadrada del error cuadrático medio, RMSE	$\sqrt{\text{MSE}}$
Coeficiente de determinación	$1 - \frac{\sum_{i=1}^{N}(d_i - \hat{d}_i)^2}{\sum_{i=1}^{N}(d_i - \bar{d})^2}; \bar{d} = \frac{1}{N}\sum_{i=1}^{N} d_i$
Error de predicción final, FPE	$\frac{p+g}{p-g}\frac{\hat{d}^T P^2 \hat{d}}{p}$
Índice de error aparente, APER	$\frac{\sum n_{i=1}^{k}}{\sum N_{i=1}^{k}}$
Índice de clasificación correcta, CCR	$1 - \text{APER}$
Criterio de información de Schwarz, BIC	$\frac{p+(ln(p)-1)\gamma}{p-\gamma}\frac{\hat{d}^T P^2 \hat{d}}{p}$

donde N representa el tamaño de la muestra, $\hat{d}_i$ son los valores ajustados y d_i son los valores muestrales. Además, $a_i = 1$ si $(d_{i+1} - d_i)(\hat{d}_{i+1} - d_i) > 0$, y 0 en caso contrario. El criterio de información de Schwarz también es conocido como Criterio de Información Bayesiana. AIC y BIC deberán ser lo menores posible, pudiendo tomar valores negativos, y ambos miden cuánto se ajusta el modelo estimado a los datos. P es la matriz de proyección, p es

el número de patrones (*patterns*), y γ es el número efectivo de parametros, $\gamma = \text{traza}(A^{-1}H^TH)$, [54].

En muchas aplicaciones, diversos modelos ajustados pueden ser adecuados en términos de los comportamientos de los residuos. Una forma de discriminar entre estos modelos competidores es utilizar los llamados criterios de información que tienen en cuenta no solo la calidad del ajuste, sino que penalizan la inclusion de parámetros extras. Así, un modelo con más parámetros puede tener un mejor ajuste, pero no necesariamente será preferible en términos de criterios de información.

Capítulo 4

Experimentos

4.1. Pivoteo QLP para identificar la dimensión

El algoritmo de pivoteo QR tiene una buena reputación como procedimiento de pivoteo para identificar las columnas importantes en una matriz de datos. Si se observa una diferencia sustancial en los valores singulares de la matriz A (digamos que se observa un salto evidente en el valor σ_m), en la diagonal del factor R se detectará asimismo una diferencia en el valor r_{mm}, aunque este otro salto puede no ser igual de sustancial.

Para simplificar la exposición, nos referiremos a los valores en la diagonal de R como R-valores, a los de la diagonal de S como S-valores y a los de la diagonal de L, de la descomposición QLP, como L-valores. El valor r_{mm}, indica el índice mm del rango numérico de la matriz A descompuesta.

La matriz A, $A_{n\times p}$, corresponde a las neuronas definidas por su centro y su radio. Esta matriz puede interpretarse como un conjunto de p neuronas (columnas) en R^n. Por ejemplo, como ilustración de la detección de un salto evidente mediante la aplicación de la descomposición QLP, generamos una matriz A de orden 100 mediante la fórmula

$$A = USV' + 0{,}1\sigma_{50}E \tag{4.1}$$

donde:

1. S es una matriz diagonal con valores geométricamente decrecientes de 1 hasta 10^{-3} con los últimos 50 valores sustituidos por 0 (de manera que se fuerza un rango exacto igual a 50).

2. U y V son matrices ortogonales.

3. E es una matriz de desviación típica normal, que se añade para generar ruido y de esta forma conseguir un rango deficiente en la matriz resultante, A.

De esta manera, A representa un matriz de rango 50 perturbada por un error cuyos elementos son del orden del último valor singular no nulo.

Hemos considerado varias opciones: en primer lugar el uso de SVD seguido de QR para determinar las columnas importantes en la matriz de partida. También hemos considerado la opción de no usar el procedimiento discutido de SVD seguido de QR, sino solamente QR. [86] sugiere usar únicamente la observación de un posible *gap* en el factor R resultante de QR. De este modo es posible obtener información a partir de dicho factor, sin usar SVD (que resulta más costoso computacionalmente).

Como tercera opción consideramos el uso de la descomposición QLP como alternativa. Esta descomposición, como veremos en los resultados, ofrece un rendimiento de detección similar al de SVD y QR, y aún requiere menor coste de cómputo.

Por tanto en un contexto de implementación práctica o en tiempo real donde hay restricciones de cómputo, podríamos prescindir del SVD y usar solamente QR o, mejor aún, QLP.

En resumen, y realizando una replicación del experimento que ilustra lo esbozado en [86]:

- Crear sendas matrices aleatorias ortogonales U y V.
- Construir una matriz, S, de 100 valores *singulares* en la diagonal, donde los primeros 50 sean decrecientes pero distintos de cero, y los 50 restantes sean exactamente cero. Se intenta comprobar si el QR es capaz de detectar esto en la matriz, A, resultante. Para intentar *despistar* al algoritmo QR, añadimos una componente de ruido aleatorio, $0{,}1\sigma_{50}E$, a la matriz USV'.
- Se determinan gráficamente las capacidades relativas de SVD (valores singulares), QR (R-valores) y QLP para determinar el rango numérico en la matriz resultante, mediante la detección de un salto en los correspondientes trazados logarítmicos.

4.1.1. Comparación de las capacidades de detección del rango numérico (*gaps*)

Aunque el SVD sea razonablemente bueno para revelar saltos en los valores singulares de una matriz, como mencionamos a la hora de discutir el

procedimiento QLP, esta detección podría mejorarse. Además, los R-valores tienden a subestimar los valores singulares grandes y sobrestimar los pequeños. En esta parte se considera la descomposición QLP que proporciona una nueva descomposición con mejor propiedad de descubrimiento de saltos muy cercana al SVD, pero con menos cómputo, como veremos posteriormente.

Implementamos en Matlab® el algoritmo de la descomposición QLP que permite determinar el salto (y por tanto el rango numérico) de una matriz A. Nuestra idea es sustituir los dos pasos, SVD+QR, primeramente en un único paso, QR, pues se demostró en el experimento de los saltos que el QR por sí solo determina de forma bastante fiable los saltos asociados al rango numérico. Sin embargo, se deduce que con la descomposición QLP se obtiene una determinación de la significación en la diagonal mucho más precisa y cercana a la que se obtenía con SVD, ver las figuras 4.1 y 4.14, y con menos cómputo, ver la tabla 4.1; por tanto, nuestro objetivo es usar en adelante la QLP en lugar de la SVD e incorporarlo en nuestro algoritmo de reducción de matrices de datos de entrada o de activaciones neuronales de las RBF.

Analizados los resultados de las tres descomposiciones mencionadas mostramos en la figura 4.1 la comparación de QLP, QR y SVD.

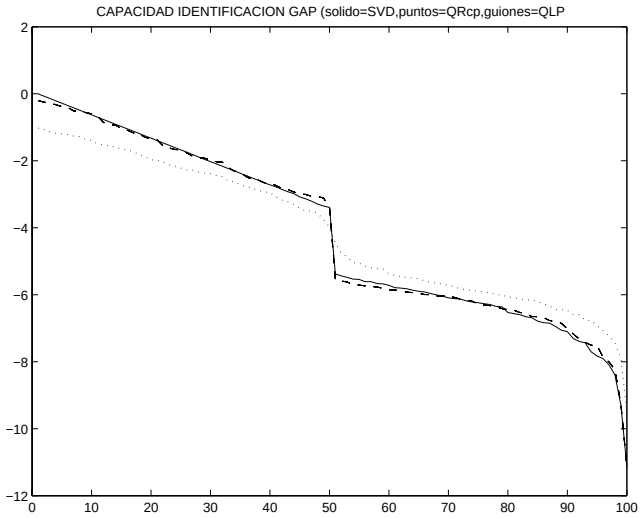


Figura 4.1: Comparaciones de las descomposiciones QLP, QR y SVD

Analizando el QLP, obtenemos que el número de condición de A, cond(A), vale 6,9129E+004; este elevado valor indica que la matriz está próxima a ser singular, o sea, el rango es menor que n, y existe al menos una columna, correspondiente a una neurona, que es combinación lineal de las demás. Esta singularidad indica que el número de neuronas o rango numérico no es

realmente 100. El rango algebraico o real de la matriz A podría ser 100, porque es una pertubación minima aleatoria de una matriz de rango exacto 50. Sería mucha casualidad que las perturbaciones introducidas diesen lugar a dependencia lineal, por tanto la función `rank` devuelve el valor 100, pero no podemos *fiarnos* cuando el valor de `cond` es tan elevado, eso indica rango deficiente, es decir, que una mínima perturbación equivalente a eliminar la perturbación aleatoria, podría hacer pasar de repente de una matriz de rango exacto 100 a una de rango exacto 50. Entonces si `cond` da un valor alto, conviene averiguar el rango numérico adecuado recurriendo a una de las funciones SVD, QR o QLP, preferentemente QLP como hemos visto, para determinar el rango numérico. El lugar donde se encuentra el *gap* o cambio súbito de significación, da una estimación del rango numérico; en este caso particular el rango numérico es 50.

Nuestro objetivo es doble: en primer lugar obtener una estimación del rango numérico, lo cual conduce a saber el número de columnas necesarias en una matriz neuronal o de datos. En segundo lugar, saber cuáles son esas columnas significativas. Para el primer paso hemos visto que se puede usar SVD, QR o QLP, siendo QR la peor opción pues ofrece una identificación más pobre. Pero el inconveniente de SVD es su elevado coste computacional, por tanto es interesante ver que usando QLP, que es nuestra sugerencia, se obtiene una precisión en la identificación similar a usar SVD, pero con la ventaja adicional de requerir menos cómputo que SVD, y además como se obtienen con QLP permutaciones de salida, resolvemos los dos pasos en un único procedimiento, es decir, tanto saber el número de columnas adecuado (rango numérico) como identificar cuáles son exactamente dichas columnas mediante la permutación.

El problema puede ser cómo determinar una permutación única que afecte a toda la matriz original colocando las columnas importantes en primer lugar como se hacía con QR, pero con las dos subpermutaciones, **pl** y **pr**, (ver la rutina del procedimiento QLP) que se obtienen con QLP, es decir, cómo unir esos dos factores para obtener una única permutación que coloque las columnas importantes al principio de A.

Observando las dos subpermutaciones no vemos una información clara, está claro que hay que combinarlas de alguna forma en una única permutación, como ocurría con QR. Observando el algoritmo de la function QLP, vemos claramente que el factor **pr** coincide con la permutación que se habría obtenido en caso de usar QR (ver el primer comando de la función en el algoritmo). Por tanto, a falta de información adicional sobre **pl**, al menos sabemos que en la permutación **pr** está el mismo resultado de permutación que se hubiera obtenido si nos hubiéramos conformado con un QR.

4.1.2. Tiempos de cálculo para las descomposiciones SVD, QR y QLP

El propósito de esta sección es demostrar que el tiempo de cómputo de las descomposiciones QR y QLP en comparación con la SVD es menor, y por tanto el algoritmo es más rápido. Como ejemplo se considera la misma ecuación del pivoteo, 4.1. En las tres descomposiciones se obtienen los tiempos de cómputo indicados en la tabla 4.1.

Descomposición	Tiempo de CPU
QR y QLP	0,01
SVD	0,03

Cuadro 4.1: Tiempo de las descomposiciones

4.2. El filtro de Wiener

Una señal, $f(x)$, ha sido contaminada por un ruido convolutivo y aditivo, obteniéndose $y(n)$. Queremos determinar el filtro lineal, FIRw_k, con el que se obtiene la mejor estimación, $\hat{x}(n)$. La reconstrución de la señal con un filtro lineal de Wiener, $\tilde{h}$, viene dado por la siguiente ecuación:

$$y(x) = (b * \tilde{h})(x) \tag{4.2}$$

donde $f(n)$ representa los datos, $y(n)$ la salida y $d(n)$ el valor deseado. Se considera que la señal, $f(n)$, está contaminada con ruido aditivo incorrelado, esto es , $\varphi_{f(n),\eta(n)} = 0$. El propósito de este filtro es reconstruir la señal, $f(x)$, más cercana posible, $b(x)$, con $d(x)$.

El objetivo de este sección es plantear y resolver el problema de diseño de una respuesta impulsional, $h(x)$, de modo que la salida, $y(x)$, sea lo más parecida posible a una señal denominada referencia, $d(x)$, figura 4.2, donde $y(n)$ es la salida del filtro y $d(n)$ es el vector objetivo, señal sin ruido, y

$$b(n) = f(n) + \eta(n) \tag{4.3}$$

es la señal original contaminada por un ruido. El valor de $\tilde{h}$ representa el filtro de la operación.

El parecido puede establecerse con cualquier criterio que se estime oportuno; no obstante, un planteamiento lineal del programa obliga a tomar,

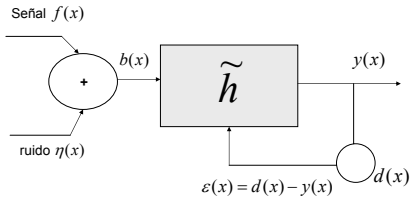


Figura 4.2: Esquema del filtro de Wiener

como medida de dicho parecido, el error cuadrático medio entre la salida y la referencia.

$$\epsilon(x) = E[d(x) - y(x)]^2 \tag{4.4}$$

Así pues, el filtro óptimo, se entiende que en términos de error cuadrático medio, es aquel que minimiza el MSE. Los coeficientes del filtro óptimo se obtendrán al derivar esta última expresión con respecto a ellos e igualar a cero. De nuevo, al tratarse de una forma cuadrática, cumple las condiciones de Cauchy-Rieman, por lo que el gradiente del objetivo con respecto al vector hermítico, al igualarse el vector a cero, proporciona la solución buscada.

$$\nabla\epsilon(x) = \underline{0} \tag{4.5}$$

En otras palabras, insertando las ecuaciones 4.3 y 4.4 en 4.5, minimizando el error a través de la derivada de 4.4, y posteriormente igualando a cero 4.5, se obtiene:

$$\tilde{h} = E\{f(x)f(x)^T\}^{-1}E\{d(x)\underline{f(x)}\} \tag{4.6}$$

donde $E\{f(x)f(x)^T\} = A_{xx}$ es la matriz de autocorrelación de los valores de entrada.

Si se define $\underline{Z} = Ed(x)\underline{f(x)}$, la ecuación 4.6 puede ser reescrita:

$$\tilde{h} = A_{xx}^{-1}\underline{Z} \tag{4.7}$$

donde el filtro, $\tilde{h}$, debe ser estimado a partir de los datos, lo que puede realizarse con la descomposición QLP en la matriz de autocorrelación, A_{xx}.

Para este cálculo se ha considerado la inversa mediante las tres descomposiciones para encontrar los coeficientes, $\tilde{h}$, del sistema de ecuaciones de Wiener-Hopf, 4.7.

Expandiendo la ecuación 4.3 se tiene que:

$$\epsilon(x) = \sum_{i=1}^{n} (f(x_i) - b(x_i)\tilde{h}_i)^2 \tag{4.8}$$

Esta expresión permite escribir 4.3 como una versión alternativa del MSE que evidencia la dependencia cuadrática del MSE en el filtrado implementado. Insertando 4.3 en 4.8, se puede escribir:

$$\epsilon(x) = \sum_{i=1}^{n} (f(x_i) - (f(x_i + \eta(x_i))\tilde{h}_i)^2 \qquad i = 1, \ldots, n \tag{4.9}$$

donde $i = 1, \ldots, n$ es el número de la señal de entrada, $f(x)$.

Después de revisar estos aspectos del filtro de Wiener, es interesante plantearlo en términos geométricos. Se puede hacer una interpretación geométrica del planteamiento y diseño del filtro de Wiener que es una ayuda inestimable a la hora de entender su funcionamiento. Se mostrará esta interpretación a continuación para el caso de dos dimensiones, es decir, $K = 2$. Si el filtro a diseñar cuenta con tan solo dos componentes, $h(0)$ y $h(1)$, los datos utilizados serán solo $x(n)$ y $x(n-1)$, y la salida, $y(n)$, será la combinación lineal de estos dos últimos según $h(0)$ y $h(1)$. Al ser $y(n)$ siempre una combinación lineal de los datos, puede decirse que estará contenida siempre en el plano de los datos, como se representa en la figura 4.3.

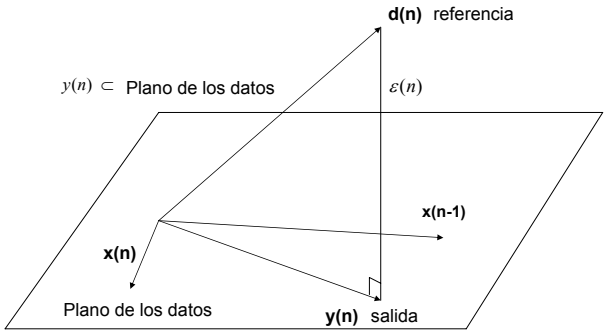


Figura 4.3: Interpretación geométrica del filtro de Wiener para orden 2

Así, nuestro objetivo es combinar el uso del filtrado de Wiener con las tres descomposiciones matriciales aplicadas en la matriz de autocorrelación

A_{xx}^{-1}, o sea $A_{xx}^{-1} = P1.(L2)^{-1}.Q2 \Rightarrow \tilde{h} = P1.(L2)^{-1}.Q2.Z$, conforme a las ecuaciones 2.38 y 4.7.

4.2.1. La relación señal-ruido

La relación señal-ruido (Signal to noise ratio, SNR) es el cociente entre la potencia de la señal, $f(x_i)$, y la potencia del ruido, $\eta(x)$ (Ver figura 4.2).

$$H(x_i) = \frac{(|f(x_i)|)^2}{(|b(x_i)|)^2} = \frac{(|f(x_i)|)^2}{(|f(x_i) + \eta(x_i)|)^2} \tag{4.10}$$

Con esta última información es posible estudiar los resultados obtenidos por simulación usando nuestra descomposición QLP en un algoritmo recursivo para encontrar los coeficientes de Wiener, a través de la matriz de Toeplitz, A_{xx}, usando la descomposición SVD y la descomposición propuesta QLP.

4.3. Filtrado de Wiener

Nuestro objetivo no es desarrollar en profundidad el filtrado de Wiener. En lugar de ello, presentaremos en esta parte, a modo ilustrativo, un experimento llevado a cabo usando la descomposición QLP, comparando su rendimiento con las descomposiciones QR y SVD. El material expuesto en la sección 4.2 será util para entender esta aplicación.

4.3.1. Modelo teórico

A lo largo de este experimento consideramos una simulación, [17], usando los coeficientes $\beta = (0{,}8; 0{,}5; -0{,}1; -0{,}3)$. La señal de entrada, $f(x)$, está formada del siguiente modo:

$$f(x) = \sum_{i=5}^{1500} \sum_{j=1}^{4} f(i-j)\beta(5-j) + \epsilon(x), \tag{4.11}$$

donde $\epsilon(x) \sim N(0, \sigma^2)$.

La señal objetivo viene dada por:

$$d(x) = \sum_{i=5}^{1500} \sum_{j=1}^{4} f(i-j)\beta(5-j), \tag{4.12}$$

En este algoritmo recursivo se ha considerado la variación del SNR en relación con el orden del filtro, y la variación del SNR en relación con el

tamaño del vector de entrada. Para verificar el tiempo en este algoritmo también se usará el tiempo de CPU en relación con el tamaño del vector de entrada y el tiempo de CPU en relación con el orden del filtro.

Los tamaños de la señal de entrada son $N = 2^{k_1}, k_1 < 11$, y los órdenes del filtro son $M = 2^{k_2}$, $k_2 < 8$. El tamaño de la matriz de Toeplitz, A_{xx}, se obtiene a partir de mín(máx(M), máx$(N)) = 128$, y por tanto este tamaño es 128×128. Se aplicó la descomposición QLP a la matriz de Toeplitz A_{xx}, la inversa de esta matriz está definida en las ecuaciones 2.37 y 2.38. Así, los coeficientes de Wiener constituyen una matriz de tamaño 1×128, de acuerdo con la ecuación 4.7.

El primer filtro fue aplicado al vector de entrada, $f(X)$, y el segundo filtro fue aplicado al ruido, η.

La relación señal-ruido, SNR, es de tamaño $500 \times 10 \times 7$, donde 500 representa el número de veces que se ejecuta este algoritmo recursivo, 10 representa el tamaño de la señal, y 7 es el orden del filtro.

El cálculo de la matriz de Toeplitz, A_{xx}, usando las tres descomposiciones muestra que el tiempo de cómputo de los algoritmos QR y QLP es 2,3 más corto que el tiempo de cómputo usando la descomposición SVD (ver tabla 4.2).

Descomposición	Tiempo de CPU
QR y QLP	0,03
SVD	0,07

Cuadro 4.2: Tiempo de cómputo de la descomposición de A_{xx}

Considérense las figuras 4.4, 4.5 y 4.6. En el ítem (a) puede observarse que presentan la misma variación del SNR cuando se relaciona con el orden del filtro. Así, las tres descomposicones presentan prácticamente el mismo resultado. El comportamiento en el ítem (b) confirma el resultado de la tabla 4.1, mostrando que las descomposiciones QR y QLP son más rápidas que la descomposición SVD, y la variación SNR para todos los tamaños de la señal usando la descomposición QLP está por debajo de 2,8, mientras que las descomposiciones SVD y QR tienen una variación un poco mayor, por debajo de 3, a partir del tamaño de entrada con índice 7. En el ítem (c) para todas las descomposiciones se verifica que el tiempo de CPU en el caso SVD fue de aproximadamente $0{,}05s$, mientras que QR y QLP tuvieron aproximadamente $0{,}04s$.

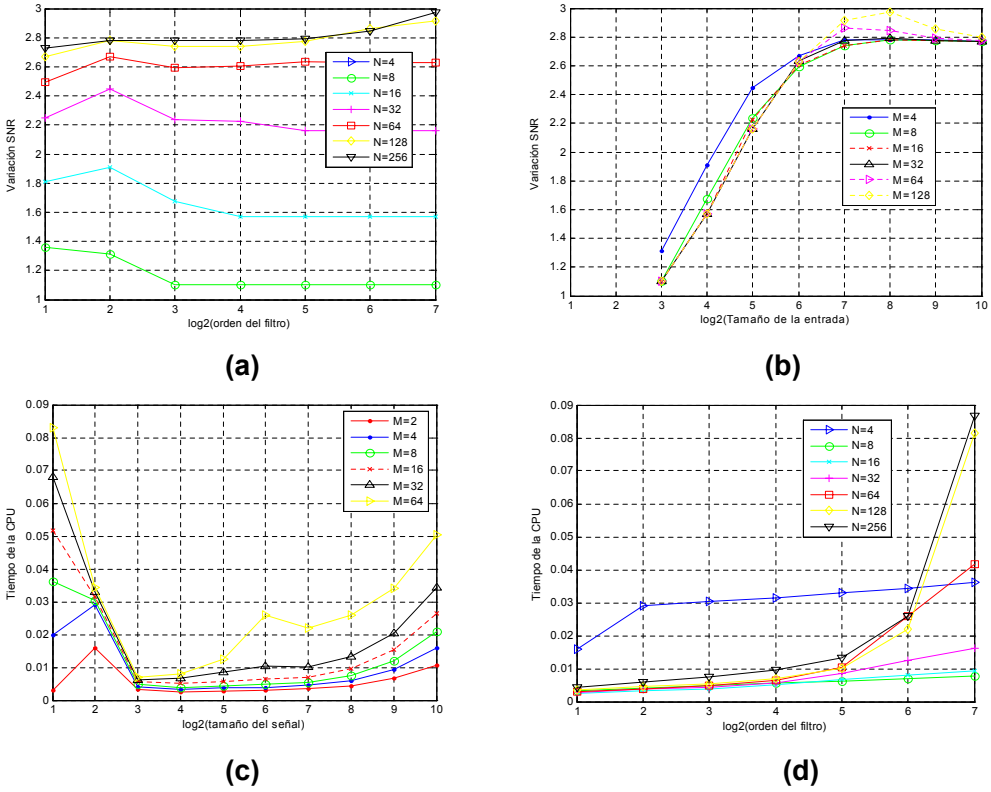


Figura 4.4: Descomposición SVD. Variación SNR (razón de tiempo de Wiener) frente al orden del filtro (a), Variación SNR (razón de tiempo de Wiener) frente al tamaño de la señal (b), Tiempo de CPU frente al tamaño de la señal (c), y Tiempo de CPU frente al tamaño del filtro (d)

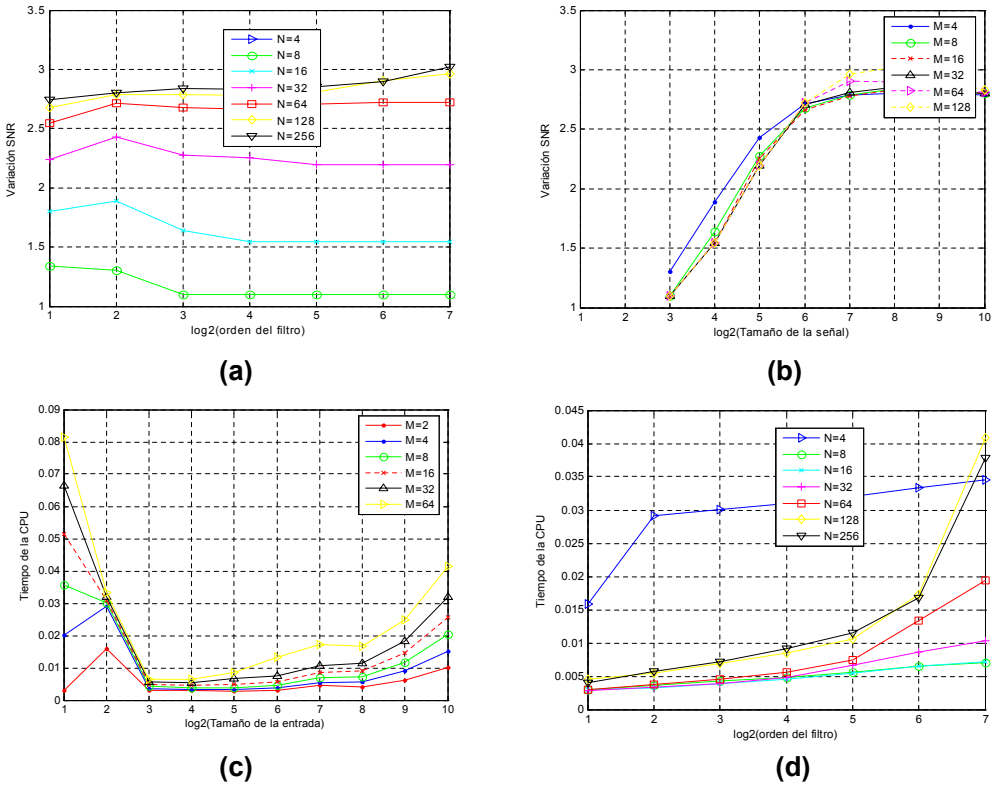


Figura 4.5: Descomposición QR. Variación SNR (razón de tiempo de Wiener) frente al orden del filtro (a), Variación SNR (razón de tiempo de Wiener) frente al tamaño de la señal (b), Tiempo de CPU frente al tamaño de la señal (c), y Tiempo de CPU frente al tamaño del filtro (d)

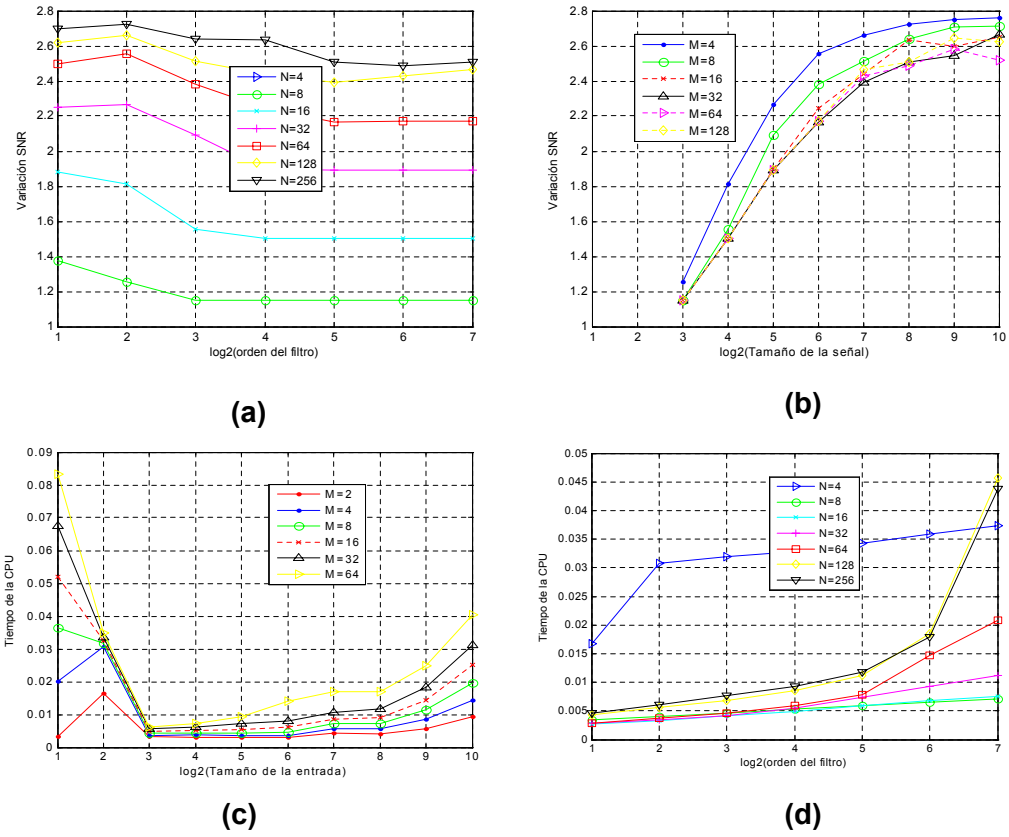


Figura 4.6: Descomposición QLP. Variación SNR (razón de tiempo de Wiener) frente al orden del filtro (a), Variación SNR (razón de tiempo de Wiener) frente al tamaño de la señal (b), Tiempo de CPU frente al tamaño de la señal (c), y Tiempo de CPU frente al tamaño del filtro (d)

4.4. Ajuste de la función SinE

La técnica presentada en esta sección permite resolver el problema de aproximación sin necesidad de recurrir a suposiciones con respecto a la forma de la función a aproximar. Si en un determinado problema se desea aproximar una función no lineal, partiendo de una serie de datos afectados con ruido, el problema mas común es realizar un regresión funcional, para ello es necesario suponer que una función determinada expresa de la mejor forma posible la relación entre las variables dependientes y las independientes. Para confirmar el tipo de función que mejor expresa esta relación existen técnicas estadísticas tales como la determinación del coeficiente de correlación. Sin embargo, el hecho de introducir esta suposición sobre el tipo de función, es un factor que limita la obtención de la mejor aproximación a un determinado conjunto de datos afectados por ruido. En general, la regresión funcional está inherentemente mal condicionada, y por tanto no existe una solución única.

En este trabajo se describe una reducción de neuronas RBF con descomposición QLP para resolver este problema de aproximación. Para esta aproximación se usarán diferentes tipos de funciones de base radial en cuanto a su naturaleza: Gaussiana, Multicuadrática y Cauchy. El primer caso que vamos a analizar será el de la Gaussiana.

4.4.1. RBF gaussiana con descomposición QLP

Como caso práctico se plantea un problema de aproximación que consiste en la función escalar $z(x) = 0{,}8\exp(-0{,}2x)\,\mathrm{sen}(10x)$ que suponemos desconocida [31].

A continuación se describen los pasos a seguir para la resolución de dicho problema utilizando las redes de neuronas de base radial.

- Conjunto de muestras o ejemplos sobre el problema. A partir de la expresión analítica de la función **SinE** se extrae un conjunto de 1100 muestras, las cuales se generan siguiendo una distribución uniforme en el intervalo $[0, 10]$.

- Extracción del conjunto de entrenamiento y comprobación. Del conjunto de muestras generadas, se utilizan 100 muestras extraídas aleatoriamente como patrones de entrenamiento y las 1000 restantes se utilizan como patrones de comprobación.

- En nuestro caso, la RBF tendrá una neurona de entrada que recibe el valor de la variable, X, y una neurona de salida. En principio se fijan

51 columnas de la matriz de diseño de acuerdo con la descomposición QLP. Posteriormente, se cambiarán dichos parámetros para ver cómo influyen en la resolución del problema.

Como hemos indicado, 100 valores serán usados para el entrenamiento, generados mediante $y = z(x) + \epsilon$, donde X se distribuye uniformemente en el intervalo $[0, 10]$ y $\epsilon \sim N(0, 1)$. Los 1000 datos de comprobación estarán aleatoriamente distribuidos en el rango $[0, 10]$. Para los conjuntos de datos de entrenamiento y comprobación fijamos una RBF Gaussiana con parámetro de regularización $\lambda, \lambda = 10^{-3}$.

La función base gaussiana se define con una desviación $\sigma^2 = 0{,}1$. La totalidad de los 100 puntos fueron usados como candidatos para el centro de la RBF Gaussiana.

En la figura 4.7 se representan los errores del modelo con 100 neuronas, que presentan una estructura aleatoria. Considerando las 100 neuronas se puede verificar que la aproximación de la RFB no es buena, ver figura 4.8. Esto prueba que si usamos todos los datos, el resultado sería una función que no serviría, ya que en realidad aproxima el ruido. En muchos casos, sin embargo, ocurre que para poder construir una aproximación mediante la suma de aproximaciones locales se requiere un alto número de neuronas ocultas, lo cual podría influir negativamente en la capacidad de generalización de las RBF.

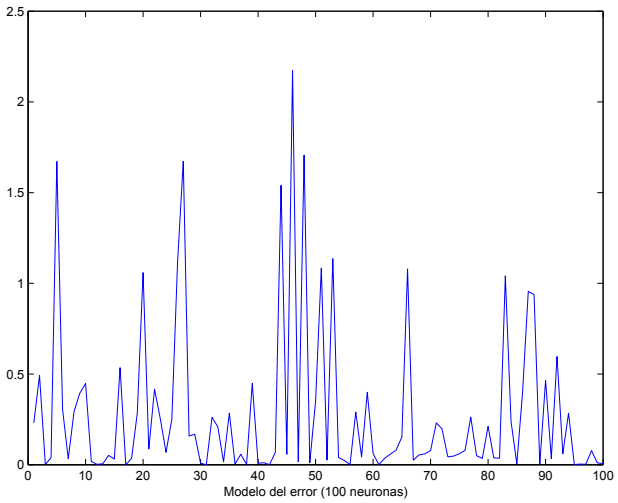


Figura 4.7: Modelo de error con 100 neuronas

En la figura 4.9 se observa que para todas las descomposiciones consideradas se han localizado aproximadamente 51 neuronas RBF diferentes de

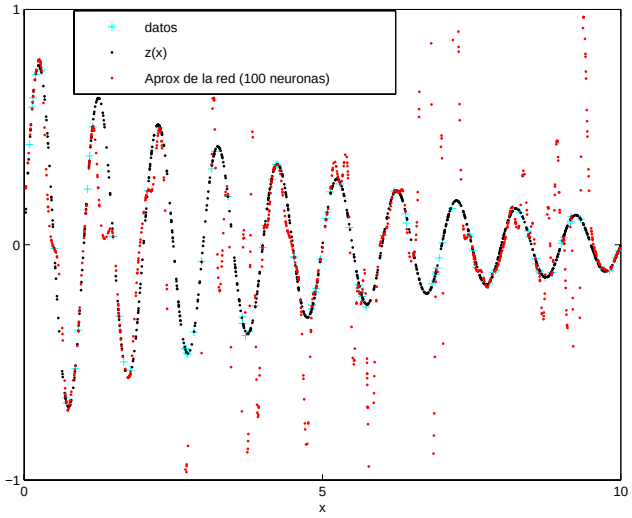


Figura 4.8: Aproximación de SinE con 100 neuronas de una red RBF

cero.

En la figura 4.10 se muestra la superposición de la función original, **SinE**, los datos de entrada contaminados con ruido, y la función determinística que la red neuronal fue capaz de aprender, o sea, la aproximación de la RBF reducida mediante QLP. En este caso se obtine una buena generalización después de la reducción de las 49 neuronas según el QLP.

Si comparamos la representación del error con 100 neuronas con la del modelo de 51 neuronas, la dispersión de los valores de la función **SinE** con respecto al modelo de la RBF gaussiana reducida presenta en este último una amplitud máxima inferior a 1,8, mientras que en el modelo de error con 100 neuronas alcanza un valor de 2,3.

En la figura 4.11 se realiza la representación de los errores del modelo estimado, que presentan una estructura aleatoria, hecho muy favorable como prueba de diagnosis del modelo de predicción RBF reducido por QLP.

Las redes de neuronas de base radial son de carácter local, ya que, dado un patrón de entrada a la red, $X(n)$, si está en la vecindad del centro, C_i, de la neurona oculta, i, esta alcanzará un valor alto de activación [44]. El parámetro c (centros) es igual a los parámetros de entrada x en la inicialización. En esta función de difícil ajuste, sería interesante considerar centros de neuronas a lo largo del eje X, una vez que la función presenta un decaimiento regular.

Un método alternativo para superar el escollo del número de neuronas es el BIC, este valor confirma aproximadamente el número de neuronas a considerar, ver tabla 4.3. Este criterio es un método para la selección de un

	BIC		FPE	
	Φ	Φ_{QLP}	Φ	Φ_{QLP}
100	0,0003	0,003	0,0001	0,0010
51	-	0,007	-	0,0040
45	-	0,010	-	0,0057
40	-	0,014	-	0,0080
35	-	0,026	-	0,0157

Cuadro 4.3: Número de neuronas de la Gaussiana, BIC y FPE

modelo. Es aconsejable escoger un modelo que tenga el mínimo valor de este estadístico. Aquí se ha seleccionado un modelo de 51 neuronas por obtener el menor BIC. Podría considerarse un segundo modelo con 45 neuronas, pues tiene un menor error de comprobación, 0,009.

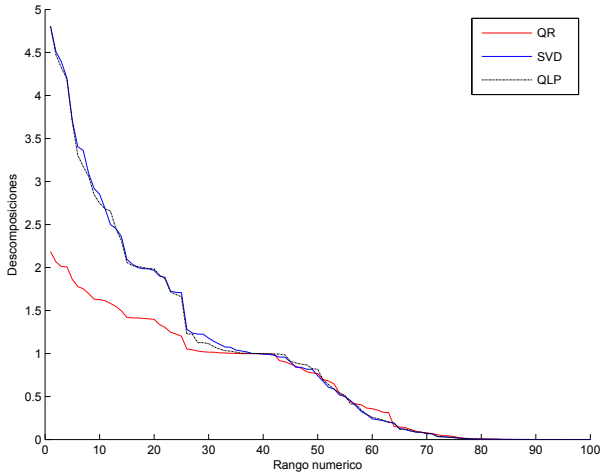


Figura 4.9: Descomposiciones SVD,QR y QLP de la matriz de diseño

A continuación, puede observarse esto mediante el gráfico 4.10. En esta figura se observan los resultados de la predicción para las 51 neuronas después del reentrenamiento. Para el ajuste con RBF reducida por QLP es mucho mejor que el modelo ajustado con 100 neuronas.

La tabla 4.4 incluye los errores de entrenamiento y comprobación obtenidos con diferentes números de neuronas RBF Gaussianas. Estos errores han disminuido en comparación con la RBF con 100 neuronas, consiguiendo por tanto una mejor precisión para el ajuste de la función **SinE**. Se observa, por un lado, que el mejor resultado se obtiene con 51 neuronas y, por otro lado,

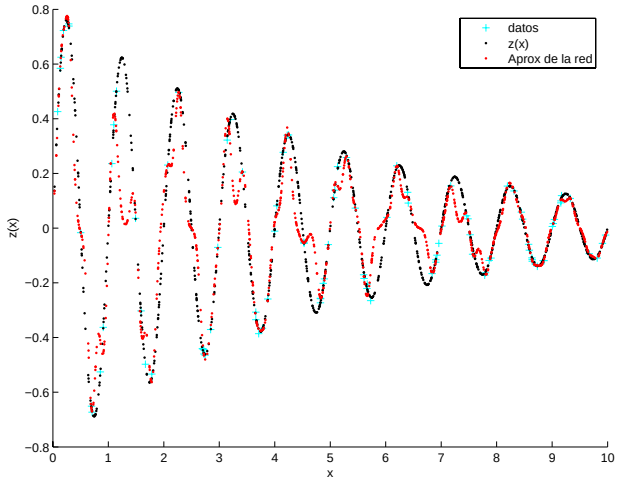


Figura 4.10: Predicción de la gaussiana con 51 neuronas RBF

que el número de neuronas ocultas en la red podría ser un parámetro significativo en la resolución del problema utilizando redes de base radiales con reducción por la descomposición QLP, pues los errores obtenidos después de la reducción son bastante diferentes.

Neuronas RBF Gauss	Error de entrenamiento	Error de comprobación
100	9,5629	2,9267
51	0,0013	0,0120
45	0,0021	0,0090
40	0,0036	0,0160
35	0,0070	0,0230
26	0,0120	0,0290

Cuadro 4.4: Errores de entrenamiento y comprobación para diferentes números de neuronas

4.4.2. RBF Cauchy con descomposición QLP

La descomposición QLP aplicada a la matriz de diseño (centros y pesos) de las neuronas RBF, ha indicado aproximadamente 25 neuronas en el caso 4.12 (a). La predicción, mediante una RBF reducida de las 25 primeras neuronas de la matriz de diseño se muestra en la figura 4.12 (b). El nuevo ajuste ha resultado el segundo mejor, después del ajuste Gaussiano.

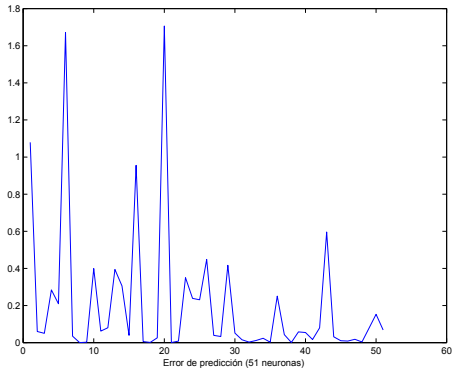


Figura 4.11: Modelo de error con 51 neuronas

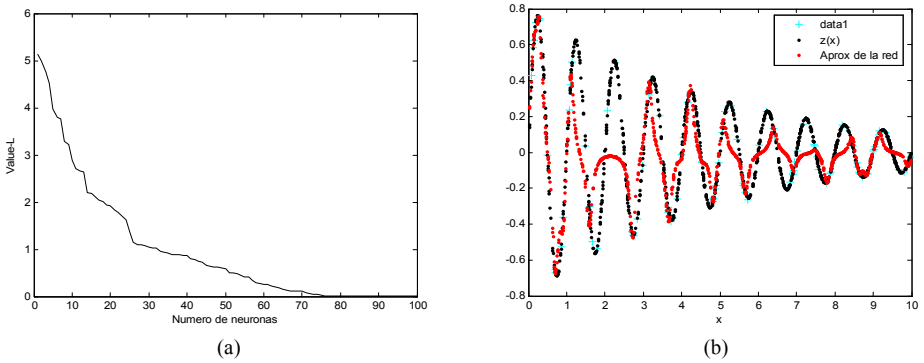


Figura 4.12: (a) Descomposición QLP de la matriz de diseño de una RBF Cauchy; (b) Aproximación por RBF con 25 neuronas

En este caso, se observa en la tabla 4.5 que los errores de entrenamiento y de comprobación obtenidos por las redes al utilizar la RBF reducida por QLP fueron de 0,012 y 0,027, respectivamente. El BIC fue de 0,030 considerando 25 neuronas, y el error de predicción final fue de 0,020, ver tabla 4.6.

Neuronas	**Error de entrenamiento**	**Error de comprobación**
30	0,010	0,024
25	0,012	0,027
20	0,026	0,040
18	0,020	0,040

Cuadro 4.5: Número de neuronas de una RBF Cauchy

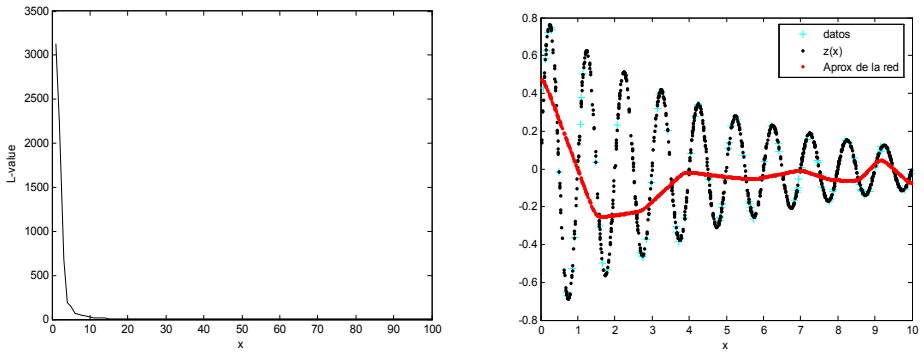
También se observa que los errores obtenidos por las redes al utilizar la RBF reducida por QLP han disminuido, con lo que se consigue una mejor precisión en los resultados. En la tabla 4.6 se indica el número óptimo de neuronas RBF tipo Cauchy de acuerdo con un menor BIC y menor error de predicción final. Esta tabla muestra que el mejor ajuste se obtiene considerando 25 neuronas, y que para 30 neuronas se alcanzaría la segunda alternativa para este ajuste.

4.4.3. RBF Multicuadráticas con descomposición QLP

En este caso, la descomposición QLP señala a un valor de 10, ver figura 4.13 (a). El modelo predictivo de la figura 4.13 (b) muestra que la aproximación no es apropiada para los patrones de comprobación. A continuación, cambiamos el número de neuronas en la red para ver si es posible mejorar los resultados obtenidos anteriormente. Se entrenan redes con 5, 8 y 12 neuronas ocultas. En la tabla 4.7 se muestran los errores de entrenamiento y comprobación cometidos para cada uno de esos números de neuronas.

	BIC		FPE	
	Φ	Φ_{QLP}	Φ	Φ_{QLP}
100	0,0003	0,003	0,0001	0,001
30	-	0,030	-	0,019
25	-	0,030	-	0,020
20	-	0,057	-	0,039
18	-	0,054	-	0,038

Cuadro 4.6: Número de neuronas de la Cauchy, BIC y FPE



(a) (b)

Figura 4.13: (a) Descomposición QLP de la matriz de diseño de una RBF multicuadrática; (b) Aproximación con 10 neuronas de una RBF multicuadrática

La tabla 4.7 muestra los errores de entrenamiento y comprobación en tanto que la tabla 4.8 presenta el BIC y el error de predicción final. El BIC permite seleccionar un modelo para este ajuste, siendo el criterio el de que se prefiere el modelo con el menor valor de dicho parámetro. En la figura 4.13 (a) se observa que aunque 10 está entre el rango de los valores indicados por el QLP, la respuesta es demasiado suave, no logrando captar la estructura de la función **SinE** y proporcionando valores bastantes alejados de los originales. Este modelo da una peor aproximación basada en el error cuadrático medio sobre el conjunto de datos de prueba. Tambien obtuvo un peor resultado en comparación con las RBF Gaussiana y Cauchy vistas anteriormente.

Neuronas	**Error de entrenamiento**	**Error de comprobación**
12	0,0570	0,071
10	0,0660	0,081
8	0,0660	0,081
5	0,0072	0,080

Cuadro 4.7: Error de entrenamiento y comprobación de una RBF multicuadrática

Destaquemos que la RBF estaba formada por todos los valores de entrada, es decir, 100 neuronas. Sin embargo, la red seleccionada según el criterio de selección usando el QLP consta solamente de 51 neuronas (caso gaussiano). Los ejemplos Gaussiano y Cauchy permiten visualizar la bondad del algoritmo para determinación del tamaño de la RBF.

	BIC		FPE	
	Φ	Φ_{QLP}	Φ	Φ_{QLP}
100	0,00024	-	0,00010	-
12	-	0,093	-	0,073
10	-	0,101	-	0,081
8	-	0,093	-	0,078
5	-	0,089	-	0,079

Cuadro 4.8: BIC y FPE de la RBF Multicuadrática con diversas neuronas

4.5. Experimento: Mapa de Hénon

El mapa de Hénon es uno de los ejemplos más ilustrativos de sistemas simples con dinámica compleja (caos determinístico). Se considera un modelo caótico con los siguientes parámetros $a = 1{,}4$ y $b = 0{,}3$ con condiciones iniciales $x_0 = 0{,}1$ e $y_0 = 0{,}9$. El factor 0,3 se incluye usualmente en la ecuación 2.52, pero la forma equivalente tiene la ventaja de que Y es el valor previo de X, o sea $Y_n = X_{n-1}$, y también se pueden escribir como: $X_{n+1} = 1 - 1{,}4X_n^2 + 0{,}3X_{n-1}$.

Para ilustrar la capacidad de la descomposición QLP para indicar los retardos más significativos en la matriz de retardos, se usará una red RBF con reducción de retardos a través de la descomposición QLP frente a una red con los retardos no significativos señalados por el QLP.

4.5.1. Resultado obtenidos con la red reducida

Consideraremos una función en Matlab® que divide la serie de Hénon, v, en dos partes: una para entrenamiento, y otra para comprobación. Después, usando la anchura de ventana de retardos, se construye una matriz de retardos, **A**, comúnmente llamada matriz de Toeplitz, de patrones de entrenamiento a partir de la porción de los retardos significativos señalados por QLP, así esta matriz, **A**, se reduce a otra matriz **P** según sus **ni** columnas más significativas (es decir, **ni** retardos significativos). Finalmente, se usa la matriz **P** y el vector **T** de respuestas deseadas correspondientes, para entrenar una red RBF. Se comcluye comparando el ajuste de la red con los retardos significativos frente a los retardos no significativos.

La predicción se realiza mediante una RBF, donde los dos primeros datos de la serie son datos de entrada de la red, y **n** será el tamaño deseado de la serie resultante.

En el algoritmo de la descomposición QLP la permutación **pr** indica las

columnas más significativas, aleatorias y reordenadas por importancia. Se extrae el número de retardos significativos a partir de la gráfica de la diagonal, L, del QLP.

Sin embargo, se deduce de [86] que con la descomposición QLP se obtiene una determinación de la significación en la diagonal mucho más precisa y cercana a la que se obtenía con SVD, figura 4.14, por tanto, nuestro objetivo es usar en adelante QLP en lugar de QR (pues esta ha mostrado casi una forma lineal cuando se ordenan los valores singulares de mayor a menor) e incorporarla en nuestros algoritmos de reducción de matrices de datos de entrada, 2.3.

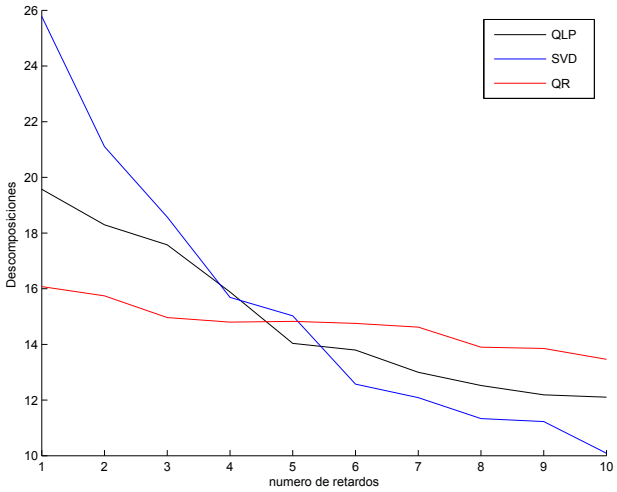


Figura 4.14: Descomposición QLP con 36 retardos

Se escogen los valores singulares de la diagonal, L, del QLP ordenados de mayor a menor, se escogen los primeros k valores de **pr**, donde $k \leq$ **pr**, inicialmente para ilustrar el aprendizaje de la RBF con los retardos más significativos según QLP, los retardos no significativos al principio se descartan, y se obtiene una matriz, P, de tamaño $n \times k$ y de rango menor, o sea $A \approx P$, donde P es la entrada de red RBF.

Como se ha dicho anteriormente, el valor **pr** del QLP revela los índices de las columnas ordenados por importancia, a saber 5 retardos para los horizontes de retardos h=7, 2, 10, 6, 5, según se ha especificado en la descomposición QLP. En primer lugar, se realiza el entrenamiento del modelo de predicción considerando los pasos de tiempos. El modelo predicho por la red tuvo 320 neuronas y la suma de cuadrados de los residuos de la red fue de 0,01. En la figura 4.15, puede apreciarse la situación al comienzo de la ejecución (iteraciones $1 - 300$).

La figura 4.16 muestra la predicción realizada por la red de neuronas para una porción del conjunto de validación. La curva continua es la serie caótica original, y la curva de puntos representa la predicción de los valores 0 a 50 de la serie caótica.

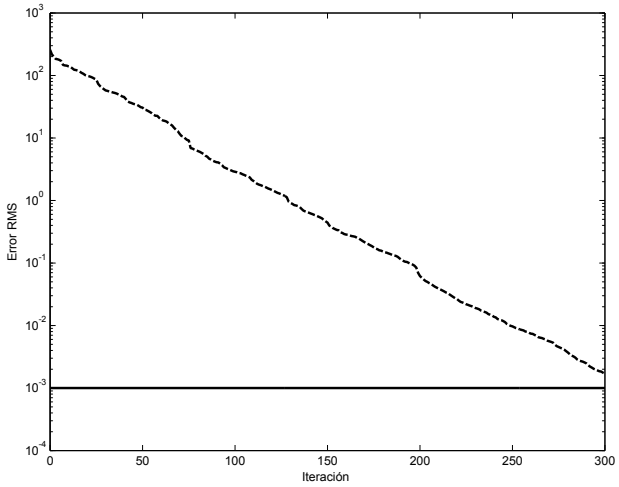


Figura 4.15: Evolución del error de aprendizaje según el número de neuronas ocultas

Para compararlas se hace la predicción de la serie de Hénon usando los 5 retardos que no fueron significativos mediante el QLP, esto es, los 5 retardos para los horizontes de retardos h=1, 9, 4, 6, 8, 3, indicados en los órdenes de 6 al 10 según el QLP. Se observa en la figura 4.17 que la predicción no tuvo bastante exactitud, o sea, que dicha predicción empeora cuando se consideran los retardos menos significativos según el QLP. En definitiva, los retardos no significativos en el método QLP resultan totalmente inadecuados para plantear la predicción de los 50 primeros valores de esta serie caótica.

4.6. Uso de una red reducida RBF para analizar y identificar dos a dos las densidades Weibull, Lognormal y Gamma

Aquí se pretende usar la RBF reducida por QLP para identificar las diferencias de estas tres densidades de probabilidad usando la reducción de neuronas dentro de la función **newrb** del entorno Matlab® . El resultado en muy heurístico, pues se ha trabajado con las funciones aleatorias **normrnd**,

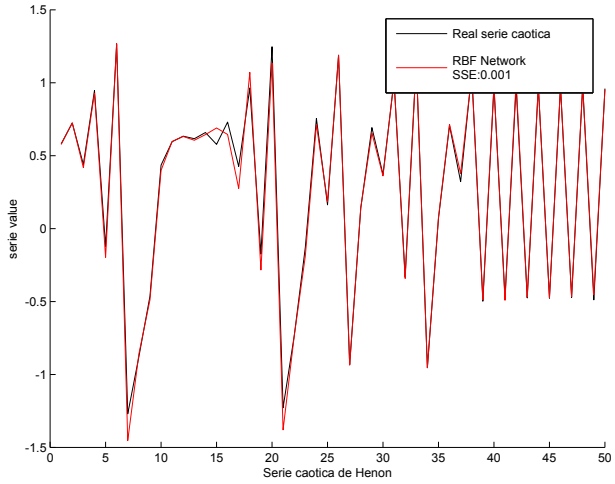


Figura 4.16: Aproximación de la serie caótica con una red RBF compuesta por los retardos más importantes según el QLP

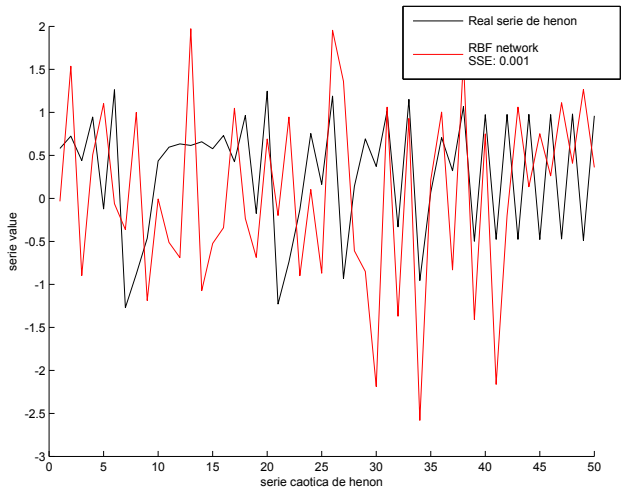


Figura 4.17: Aproximación de la serie caótica con una red RBF compuesta de los retardos menos importantes según el QLP

lognrnd y **gamrnd** del entorno citado. La pregunta que nos planteamos es: ¿Cuál es el numero mínimo de neuronas RBF reducidas por QLP que diferencian entre estas tres densidades, Weibull, Lognormal y Gamma, para las que los contrastes estadísticos no han mostrado diferencias?

Según [77] la diferencia entre las tres densidades se debe únicamente al comportamiento de la cola a la derecha.

En la figura 2.7, ilustramos estas densidades con los parámetros particulares reflejados en la siguiente tabla, pudiendo apreciarse en dicha gráfica la similitud entre las tres densidades.

Densidad	Parámetro λ	Parámetro α
Lognormal	$-0{,}0999$	0,474
Weibull	0,776	2,1
Gamma	4	0,25

El estadístico de Kolmogorov-Smirnov indica que se acepta la hipótesis de igualdad al nivel de 0,05, ver tabla siguiente, o sea, que estas tres densidades son tan similares (pese a ser diferentes) que resulta difícil la identificación a través de la estadística clásica.

Densidades	P valor
Weibull $\leftrightarrow$ Gamma	0,3124
Lognormal $\leftrightarrow$ Gamma	0,3766
Weibull $\leftrightarrow$ Lognormal	0,1040

El énfasis está en la identificación de estas tres densidades tan similares a través sus características descriptivas usando reducción de neuronas en la capa oculta de una RBF. Para eso se utilizará la descomposición QLP para:

- Indicar las principales estadísticas descriptivas, ver tabla 4.6.
- Indicar las neuronas en la capa oculta de una RBF.

Inicialmente se crea una matriz de entrenamiento y una matriz de comprobación. La matriz de entrenamiento para cada densidad será de tamaño 200×1 y se repetirá 500 veces. El objetivo de esta repetición es encontrar los ocho descriptores estadísticos en cada línea de esta matriz. De forma análoga se actuará para la matriz de comprobación, pero considerando su tamaño de 300×1. La generación de las densidades produce dos matrices de dimensiones 200×500, por ejemplo la Gamma de tamaño 200 se repitirá 500 veces con el objetivo que la red aprenda las características descriptivas asociadas. La matriz resultante, P, tendrá dimensión 1000×8, 1000 de las densidades y 8

de las características descriptivas. Inicialmente creamos 1000 datos a través del comando **randperm**, que ofrece una permutación aleatoria de enteros, para evitar posteriormente un sesgo en el aprendizaje. Así se obtienen 1000 muestras, siendo 500 muestras de una Gamma, y 500 de una Weibull o de una Lognormal.

Tomados las densidades dos a dos, la entrada de la red RBF será una matriz de tamaño 1000×8 y el objetivo será una matriz de tamaño 1000×1 formada por números 0 y 1 que representan cada densidad respectivamente. Así, la red RBF aprenderá según las características descriptivas asociadas a cada densidad.

El QLP indicará cuales de los 8 descriptores tiene más información para este análisis y también indicará qué neuronas RBF son más relevantes [18].

Aquí se usará la reducción de neuronas conjuntamente con la reducción de entradas, no se hará un estudio detallado de la reducción de entradas, pues nuestro objetivo central es analizar el comportamiento de la red con reducción de neuronas RBF. En otras palabras estudiar estas características a través de reducción de entradas y neuronas utilizando la descomposición QLP.

La lista de las características para la reducción de entradas y neuronas se encuentra en la tabla 4.6.

Estadística descriptiva	**Simbología**
Media	mean(X)
Mediana	percentile(X,.5)
Q1	percentile(X,.25)
Q3	percentile(X,.75)
Rango Intercuartílico	iqr(X)
Desviación estándar	sqrt(var(X))
Aplastamiento	Kurtosis(X)
Asimetría	Skewness(X)

Así, conociendo las caracteristicas relevantes y las principales neuronas RBF se inicia el *pruning* de las neuronas RBF.

Gamma	Weibull	Lognormal
0,99	0,99	1,07
0,88	0,96	0,98
0,12	0,08	0,22
0,18	0,09	0,30
0,67	0,73	0,67
0,52	0,48	0,49
3,70	2,58	3,49
0,87	0,36	0,82

4.6.1. Identificación con una red reducida de los estadísticos entre Weibull y Lognormal

Usamos una red neuronal para realizar la identificación de las características descriptivas, por ejemplo: $H_a : X \mapsto$ Lognormal-característica(λ, α) frente a $H_b : X \mapsto$ Weibull-característica(λ, α); si la salida de la neurona es 0 será una Weibull, si es 1 será una Lognormal. Como vimos anteriormente, el QLP permite identificar las características descriptivas más significativas entre estas densidades, figura 4.18, de acuerdo con la cual serán consideradas 4 características descriptivas. El QLP indica el orden de los estadísticos, en nuestro caso el vector pr=(7,1,8,5,6,3,2,4) indica que la kurtosis, (7), la media, (1), la asimetría, (8), y el rango intercuartílico, (5), son los estadísticos más importantes para diferenciar estas densidades.

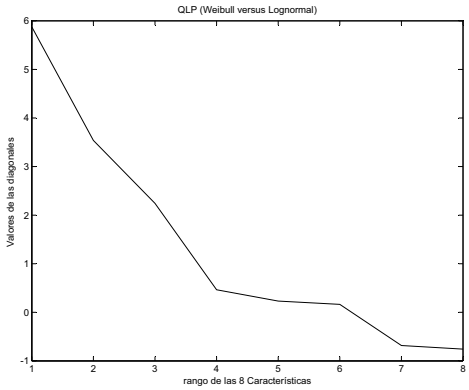


Figura 4.18: Descomposición QLP para indicar los descriptores estadísticos más relevantes entre Weibull y Lognormal

Se puede verificar en la tabla 4.6 y también en la figura 2.7 que la Weibull tiene menor kurtosis que la Lognormal. El resultado del QLP también indica

que el el rango intercuartílico es más significativo que los propios cuartiles, Q_1 y Q_2.

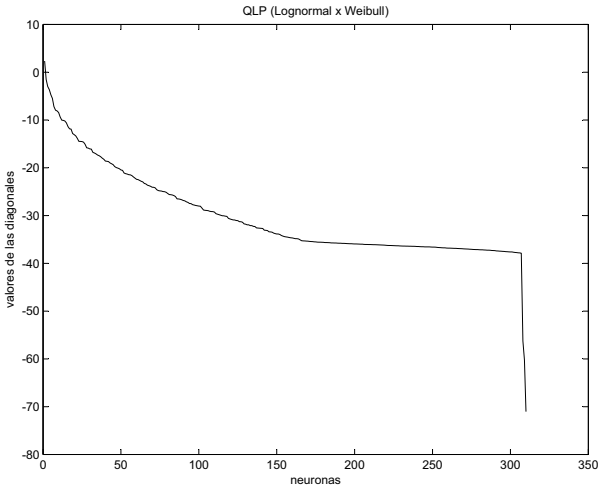


Figura 4.19: Número de neuronas suficiente para clasificar Weibull y Lognormal según el QLP

A título ilustrativo, indicamos que los índices de las 7 neuronas principales indicadas por la descomposición QLP para diferenciar las densidades Weibull y Lognormal, son 154, 104, 577, 641, 310, 8 y 663. El entrenamiento de la red con estas dos densidades fueron realizadas mediante la función **newrb**. El número de neuronas fueron 710 y se obtuvo un error cuadrático medio de 0,11. Redujimos esta gran cantidad de neuronas y también las entradas que no fueron significativas según los valores de la diagonal del QLP, ver figuras 4.18 y 4.19.

Así, ¿cúal el comportamiento de la red ante la reducción de entradas y neuronas? Para eso se crea una matriz de comprobación de tamaño 300×200, para verificar la eficiencia del *pruning* RBF (eliminación de las neuronas no relevantes a través del QLP) y la reducción de entradas (características descriptivas). Se pueden considerar varios valores, si observamos la figura 4.19, el menor número de neuronas que tiene un porcentaje de acierto superior al 70 % fue de 15 neuronas con 4 entradas. Realizada esta reducción de neuronas y también de entradas se obtienen los valores del Índice de clasificación correcta, CCR, de 0,96 tanto para la Weibull como para la Lognormal, como se indica en la tabla 4.9.

Caso	Red y entrada reducida				
Núm. entradas	8	4	4	4	4
Núm. neuronas	710	50	30	15	7
300 Weibull	295(0,98)	292(0,97)	291(0,97)	289(0,96)	267(0,80)
300 Lognormal	296(0,98)	293(0,97)	294(0,98)	290(0,96)	268(0,80)

Cuadro 4.9: Índice de clasificación correcta, para la Weibull y la Lognormal

4.6.2. Identificación entre Lognormal y Gamma

De forma análoga al anterior, en el caso de la Gamma y la Lognormal, las características descriptivas mas significativas correspondieron al vector pr=(7,8,5,2,6). Por tanto, en este caso son la kurtosis, la asimetría, el rango intercuartílico y la mediana las características más significativas, ver tabla 4.6.

Caso	Red y entrada reducida				
Núm. entradas	8	4	4	4	4
Núm. neuronas	300	300	100	60	35
300 Gamma	274(0,91)	254(0,85)	253(0,84)	253(0,84)	250(0,83)
300 Lognormal	250(0,81)	247(0,82)	247(0,82)	244(0,81)	240(0,81)

Cuadro 4.10: Índice de clasificación correcta, para la Gamma y la Lognormal

Haciendo el entrenamiento de la red hubo un total de 876 neuronas, la suma de cuadrados de los errores fue de 0,1006. Usando la matriz de comprobación se puede observar que la red con 60 neuronas y 4 entradas descriptivas, ha identificado bien la Gamma con un Índice de clasificación correcta, CCR, de 0,76, y la Lognomal con 0,96.

4.6.3. Identificación entre Weibull y Gamma

Para el caso de la Weibull y la Gamma fue preciso un elevado número de neuronas, aproximadamente 400, para poder diferenciar entre ambas. El QLP ha identificado cinco características relevantes: Kurtosis, Media, Asimetría, Rango intercuartílico y Primer cuartil, Q_1. Esto revela una información interesante respecto de estas dos densidades. En primer lugar, en la tabla 4.6 se verifica que hubo valores similares en los descriptores, en especial los descriptores Kurtosis, Asimetría y Rango intercuartílico, que fueron las

características mas importantes en los análisis de las densidades en los dos casos anteriores, Weibull-Lognormal y Gamma-Lognormal.

Esta similaridad entre los descriptores ha generado una gran cantidad de neuronas, 400, para que la red fuera capaz de discernir entre las dos densidades. Consecuentemente en el aprendizaje se necesitó una cantidad de neuronas elevada, 976, con una suma de cuadrados de errores de 0,10, esto justifica el elevado tiempo de aprendizaje. Con la reducción de 976 a 400 neuronas todavía hubo un Índice de clasificación correcta, CCR, razonable, de 0.76 para la Gamma y 0.60 para la Weibull, ver tabla 4.11.

Caso	Red y entrada reducida			
Núm. entradas	8	4	4	4
Núm. neuronas	993	400	300	100
300 Gamma	264(0,88)	230(0,76)	117(0,39)	22(0,07)
300 Weibull	287(0,95)	179(0,60)	255(0,85)	39(0,13)

Cuadro 4.11: Índice de clasificación correcta, para la Gamma y la Weibull

4.6.4. Conclusiones

Concluyendo esta sección se ha podido verificar que el método QLP es eficiente para la reducción de neuronas (*pruning*) en la capa oculta de una RBF para algunos análisis estadísticos. El resultado de la reducción ha mostrado que el Aplastamiento, la Asimetría, el Rango intercuartílico y la Media son los principales descriptores que diferencian estas tres densidades. La Weibull frente a la Lognormal con 7 neuronas y 4 entradas obtuvo los mejores resultados. En el caso de la Gamma y Weibull se necesita un análisis más profundo tanto en la entrada como en la reducción de neuronas. Para el primer caso, el método QLP es eficiente para la reducción de neuronas en la capa oculta de una RBF para identificación de características en funciones de densidad donde aparecen dificultad de identificación considerando parámetros particulares.

Capítulo 5

Differentiating distributions through RBF network pruning with QLP decomposition

5.1. Differentiating features for the F distributions with different degrees of freedom through RBF network pruning with QLP

In this thesis we propose an artificial neural network RBF to classification using feature descriptors. The theoretical and practical aspects of theory F distributions with different degrees of freedom are introduced. The distribution F densities are similar in shape, making it difficult to identify the differences between the two densities. This paper is concerned with separating these same probability densities with different degrees of freedom using feature descriptors, identified by pruning a Radial Basis Function (RBF) network using pivoted QLP decomposition generated for densities function, and its validity were evaluated by the rate of correct classification. The QLP method proves efficient for reducing the network size by pruning hidden nodes, resulting is a parsimonious model which identifies four main features (namely kurtosis and skewness and mean). The classification model induced by the methodology shows, in general, good results.

5.1.1. Introduction

Recently, [21], the Radial Basis Function network using pruning with pivoted QLP decomposition have been studied by researchers [16, 17] that identified three densities function, very simile in shape, although RBF reduction by QLP decomposition. This paper applies this model to the discrimination, from data, between two F distribution with numerator degrees of freedom V1 and denominator degrees of freedom V2. Several results show that RBF reduced by QLP decomposition are very powerful in classification and adjust [17]. The characteristic features and number of neurons RBF resulting from the network decomposition using QLP (a lower diagonal matrix L between orthogonal matrices Q and P) [87] for this particular choice of density functions has a very interesting interpretation. The distributions of interest are $F(10, 9)$ and $F(9, 8)$ density functions within these particular degrees of freedom designed. The analysis presented in this paper identifies four key discriminate features (kurtosis and skewness and mean).

5.1.2. Probability density function F

Named for the great statistician R.A.Fisher, the statistic F distribution has a natural relationship with the chi-square distribution. If χ_1 and χ_2 are both chi-square with 1 and 2 degrees of freedom respectively, then the statistic F below is F distributed. The degrees of freedom is a set of observations in a given context, the number of values that can be assigned freely, without restriction.

$$F(\nu_1, \nu_2) = \frac{\chi_1/\nu_1}{\chi_2/\nu_2} \tag{5.1}$$

The two parameters, χ_1 and χ_2, are the numerator and denominator degrees of freedom. That is, 1 and 2 are the number of independent pieces of information used to calculate 1 and 2, respectively [59]. The pdf for the F distribution is

$$y = f(x|v_1, v_2) = \frac{\Gamma\left(\frac{v_1+v_2}{2}\right)}{\Gamma\left(\frac{v_1}{2}\right)\Gamma\left(\frac{v_2}{2}\right)} \left(\frac{v_1}{v_2}\right)^{\frac{v_1}{2}} \frac{x^{\frac{v_1-2}{2}}}{\left(1+\left(\frac{v_1}{v_2}\right)x\right)^{\frac{v_1+v_2}{2}}} \tag{5.2}$$

where $\Gamma(\cdot)$ is the Gamma function.

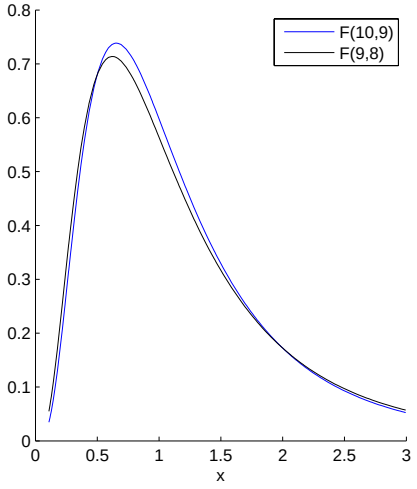


Figura 5.1: The F distribution exists on the positive real numbers and is skewed to the right

5.1.3. Detection of the Numerical rank of the QLP

The algorithm to compute the QLP decomposition can be used as an alternative to SVD and QR [87]. Since the gaps and singular subspace are defined in terms of the singular value decomposition, the natural way to compute them is to compute the singular value decomposition. Unfortunately, the computation of this decomposition is expensive. For this reason, researchers have proposed many alternatives. Of these the pivoted QR decomposition is widely recommended because of its simplicity. The pivoted QR decomposition has certain drawbacks: it gives only fuzzy approximations to the singular values, and fails to provide orthonormal bases for some of the fundamental subspaces [87, 16].

It is clear that Singular value decomposition may be easily avoided by computing the $rank - r$ principal subspace of

$$A_{xx}^{-1} = R_{xy} R_{xx}^{-1} R_{yx} \tag{5.3}$$

But the product of these three matrices, involves additional computational cost and this is made worse by the calculation of the inverse matrix. The QR decomposition produces an upper triangular matrix R_{xx}^{-1}. The QR decomposition produces an upper triangular matrix R of the same dimension as A and a unitary matrix Q so that $A_{xx} = QR$, where

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \quad . \tag{5.4}$$

R_1 is an upper triangular matrix. The diagonal elements are called the R-values of. The column permutation E is chosen so that is monotonically decreasing, $A_{xx}(:, E) = QR$. To motivate the decomposition QLP consider the partitioned $R - factor$.

$$R = \begin{pmatrix} k_{11} & k_{11}^T \\ 0 & R_{22} \end{pmatrix} \quad . \tag{5.5}$$

of the pivoted QR decomposition

$$Q^T A_{xx} \pi_R = \begin{pmatrix} R \\ 0 \end{pmatrix} \quad . \tag{5.6}$$

We can observed r_{11} is an underestimate of $\|X\|_2$. A better estimate is the norm $\ell_{11} = \sqrt{k_{11}^2 + k_{12}^T k_{12}}$ of the first row R. We can calculate that norm by postmultiplying R by a Householder transformation H_1 that reduces the first row of R to a multiple of e_1 [87]:

$$RH_1 = \begin{pmatrix} \ell_{11} & 0 \\ \ell_{12} & \hat{R}_{22} \end{pmatrix} \quad . \tag{5.7}$$

we can obtain an even better value if we interchange the largest row of R with the first:

$$\pi_1 RH_1 = \begin{pmatrix} \ell_{11} & 0 \\ \ell_{12} & \hat{R}_{22} \end{pmatrix} \quad . \tag{5.8}$$

Now if we transpose, we see that it is the first step of pivoted Householder traingularization applied to RT. If we continue his reduction and transpose the result, we obtain, [87], a triangular decomposition of the form

$$\pi_1^T Q^T A_{xx} \pi_R P = \begin{pmatrix} L \\ 0 \end{pmatrix} \quad . \tag{5.9}$$

We will call this the pivoted QLP decomposition of A_{xx} and will call the diagonal elements of L the L-values of A_{xx}.

The computation of R and L can be interleaved, so that the computation can be terminated at any suitable point, which makes the decomposition especially suitable for low-rank determination problems. We will call the diagonals of R the R-values of A_{xx}. The folklore has it that the R-values track the singular values well enough to expose gaps in the latter. For example, a matrix A_{xx} of order 100 was generated in the form [87]

$$A_{xx} = U \sum V^T + 0{,}1\sigma_{50}E \tag{5.10}$$

where $\sum$ is formed by creating a diagonal matrix (of size 1×100) decreasing geometrically from one to 10-3 and setting the last fifty diagonal elements to zero. U and V are random orthogonal matrices of size 100×100. E is a matrix of standard normal deviates. Thus A represents a matrix of rank 50 perturbed by an error whose elements are one-tenth the size the size of the last nonzero singular value. In figure 4.1 values of $k_{50,50}$ and $k_{51,51}$ show that there is a well-market gap in the $R - values$, though not as marked as the gap in the singular values [16] and [87].

The gap in the $L - value$ of the decomposition provides orthogonal bases of analogues of row, column, and null space provided by A_{xx}.

The implementation of the Matlab® package permits the QLP,

$$\text{QLP}[P, Q, L, pr, pl] = qlp(A_{xx})$$

to determine the numerical rank of matrix A_{xx}.

Thus, we have a simple QLP algorithm as follows:

1. define matrix A, which consists of the hidden node activations in the RBF network.
2. calculate the orthogonal matrices Q and P which reduce the matrix A to lower diagonal form
3. identify the diagonal of lower-triangular matrix L
4. sort the diagonal elements by size

The set of input features used in this analysis are shown in table 4.6 [51]

5.1.4. Designing RBF Neural Classifiers

Figure 2.2 depicts the architecture for a fully connected RBF network. The network consists of n input features x, M hidden units with center C_j and y output. The θ_j are the basis functions, and w_{kj} are the output layer weights. The basis function activations are then calculated using a method which depends on the nature of the function. We shall write the RBF network mapping as stated in equation 2.11.

Proposed reduction RBF for the identification of important covariates

The RBF neural network is applied here to distinguish between two density functions F in pairs, for example discriminating between the $F(10, 9)$ versus, the $F(9, 8)$. The first process is the creation of the matrix P and vector T (target matrix) of training for the problem. Each distribution has a matrix of size 200×500 comprising 500 samples of the distribution, each containing 200 observations, where x_i are uniformly randomly distributed in the $(0, 10)$. 8 features are extracted from each sample of 200 points, the features being described in table 4.6. The sample of 200×1 reduces the data matrix to 8×1 for each distribution, resulting in a data matrix of size 8×1000 of the two distributions. The training data matrix uses 8 features calculated from each column of the matrix, resulting in a data matrix of size 8×1000 when the data from the two distributions is concatenated. RBF functions are trained to distinguish between these two distributions. Starting with the number of hidden nodes (i.e. radial basis functions) equal to the number of data points, the hidden nodes are reduced using QLP decomposition. The algorithm is based in the reduction of number hidden nodes with QLP.

For each trial, a training set (x_i, p_i) was established, where p is an indicator function representing the density function which generated the sample vector x. An outrun of sample test set was generated for performance estimation. The data were generated in a similar way to the training data, but with 300 instead of 500 samples [16] and [17].

The QLP algorithm is applied twice: first, to the training data matrix, of size 8×1000, to identify the most important inputs, which are features of the distribution types; then, again, to the matrix of output node weights, of size number of hidden nodes times number of output nodes, this time to identify the most important hidden nodes, i.e. radial basis functions. The key variables identified by the QLP method are those with the highest ranking values of the diagonal matrix L. The reduction in the number of hidden nodes is carried out by sorting the diagonal elements generated by the QLP decomposition and selecting the corresponding nodes so as to give good classification accuracy with a small number of hidden nodes. The results of network training are analyzed for various conditions used in the learning process. Thus, for RBF models, the number of hidden neurons is regarded as the result of training (which depends on two key parameters, namely mean squared error goal and spread). Note that of the results reported here, the reported computational effort does not include the centre selection phase and is solely for the training of the RBF with reduction neurons. The algorithm was the same the whole analysis and the stopping criterion for training was that the error

function should be less than 10^{-2} and a maximum value of 0,5 for the radial function widths.

Apparent error rate (APER) and Correct classification rate (CCR)

We find that the overall performance of the RBF reductions classification approach is fairly good. A good classification procedure should result in few misclassifications. The apparent error rate (Aper) is the fraction of observations in the test set that are misclassified by RBF reductions (See table 3.6)

5.1.5. Experimental Results

The QLP shown in figure has 4 o 5 significant inputs, which are skewness, kurtosis, mean, deviation standard were the principal features between $F(10, 9)$ and $F(9, 8)$ densities.

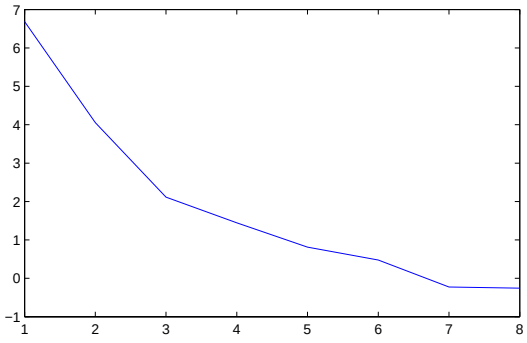


Figura 5.2: Decomposition QLP between F(10,9) and F(9,8)

A preliminary calculation of the mean values of each feature, for each distribution, is shows in the table 5.1. In this Table note that the kurtosis and skewness presents best different between these two densities F. Thus justifying that the differences between the $F(10, 9)$ and $F(9, 8)$ become most significant in the kurtosis behavior. Also shows that the sensible differences between the $F(10, 9)$ and $F(9, 8)$ aren't in the tail (see equation 5.2).

The training stopped after 975 iterations because the validation error increased. It is a useful diagnostic tool to plot the training. In the figure 5.3 shows us that the RBF had learned these two F densities with a sum squared error value of 0,0029 and total neurons of 978.

Table 5.2 shows us that of 300 values $F(10, 9)$ and 300 values of $F(9, 8)$, the RBF reduction with 200 neurons has detected a total of 174 value for

Descriptive	Density F(10,9)	Density F(9,8)
Mean	1.29	1.34
Median	1.07	1.02
Q1	0.25	0.13
Q2	0.25	0.14
iqr	0.86	1.05
sqrt	0.84	1.26
kurtosis	8.08	13.2
skewness	1.96	3.54

Cuadro 5.1: Descriptive characteristics of the probability density functions for each F density

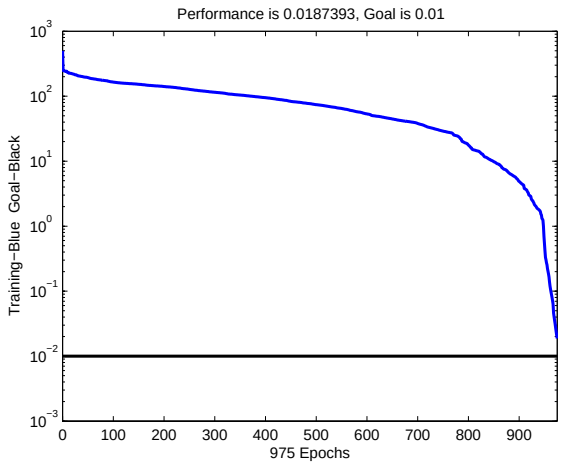


Figura 5.3: Training of radial basis function

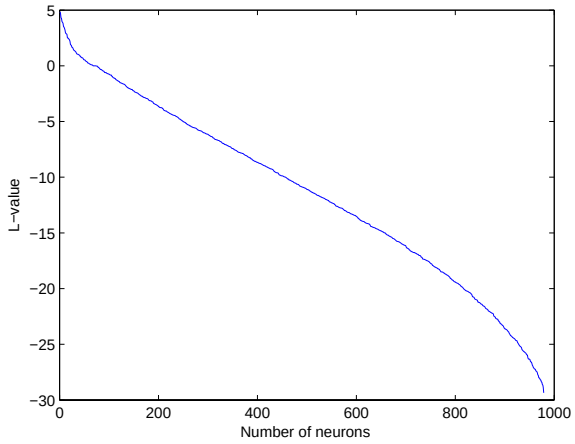


Figura 5.4: Decomposition QLP for identification number neurons RBF

the $F(10, 9)$ and 187 for the $F(9, 8)$. In this case the reduced RBF (200 neurons) has better identification between these two densities highlighted by the characteristic statistics. The apparent error rates were 0,58 for $F(10, 9)$ and 0,62 for $F(9, 8)$. In the case 8 inputs the 200 neurons RBF has identified a value of 52 % for $F(10, 9)$ and 79 % for $F(9, 8)$. The apparent error rate (APER) is represented between brackets in the following table.

Case	Number inputs	Number neural	300 (F(10,9))	300 (F(9,8))
	8	200	158(9.52)	238(0.79)
	6	100	168(.56)	178(.59)
	6	130	172(.57)	179(.59)
	6	150	173(.58)	181(.60)
	6	180	173(.58)	187(.62)
	6	*200*	*174(.58)*	*187(.62)*
	5	100	167(.55)	180(.60)
	5	150	170(.56)	187(.62)
	5	180	171(.58)	188(.62)

Cuadro 5.2: Results of RBF reduced of relationship for F(10,9) and F(9,8)

The reduced RBF by QLP decomposition identified these differences with approximately 200 neurons and 6 inputs because the curve of the frequency $F(10, 9)$ is bell-shaped and broadly similar to the $F(9, 8)$ distribution.

5.1.6. Conclusion

In this paper, a simple idea of using RBF reduction to identify characteristics among two densities has been developed. The resulting reduced networks show us that the kurtosis, skewness and mean were the principal characteristics that accurately separated the two densities. The RBF reduction by QLP decomposition can be effectively used for identification purposes. The F(10,9) and F(9,8) show us that 200 neurons with 6 inputs the RBF reduction identified a value of $CCR = 0{,}48$ to $F(10, 9)$ and $CCR = 0{,}38$ to $F(9, 8)$. The differentiation between these two distributions was less successful to below of 100 neurons. We conclude that specific class assignments to the F probability distributions with similar density functions can be accurately carried using an RBF neural network with QLP decomposition.

5.2. Using RBF reduced by QLP decomposition for Probability Density Estimation

This part is intended to be a simple example illustrating some of the capabilities of Radial basis function by pruning with QLP decomposition. The applicability of the radial basis function (RBF) type function of artificial neural networks (ANNS) approach for re-estimate the Box, Triangle, Epanechnikov and Normal densities. We propose an application of QLP decomposition model to reduce to the class of RBF neural models for improving performance in contexts of density estimate. Has been found in the QLP that such a coupling leads to more precise extraction of the relevant information, even when using it in a heuristic way. This paper is concerned with re-estimation these four densities estimated by pruning a Radial Basis Function network using pivoted QLP decomposition. For comparison all RBF type functions with the same Gaussian mixture model as the sample data is superimposed on the plot. This application tool can be used to identify the density estimate from empirical data where presents many type density estimative. The QLP methods proves efficient for reducing the network size by pruning hidden nodes, resulting is a parsimonious model which identify RBF type multiquadric to re-estimate kernel function Box and Normal distributions [19].

5.2.1. Introduction

Scott [83] shows that as the number of histograms m approaches infinity, the averaged shifted histogram becomes a kernel estimate of the probability density function. In [23] introduced the basic algorithm of nonparame-

tric density estimation. Estimating probability density functions is required in many areas of computational statistics. Another application where probability density estimation is used is in statistical pattern recognition. In other applications, we might need to determine the probability that a random variable will fall within some interval, so we would need to evaluate the cumulative distribution function The first published paper describing nonparametric probability density estimation was by Rosenblatt [80], where he described the general kernel estimator. Many papers that expanded the theory followed soon after. They addressed the problem of statistical discrimination when the parametric form of the sampling density was not known. In this paper we show how RBFs with reduction neuron thought the network decomposition using QLP (a lower diagonal matrix L between orthogonal matrices Q and P [87] and [86] using the different basis functions networks Cauchy and multiquadric, and Inverse multiquadric type function. This can resulting an approximation of the densities estimates Box and Triangle, and Epanechnikov. The performance of the RBF reduction with QLP is compared with model selection criteria as the Schwartz Bayesian Information Criterion (BIC) and mean squared error.

5.2.2. Kernel Density estimate

The estimated distribution function is calculated for a number of equidistant points that cover the range of the sample data. For each point p, the estimated density depends on the closeness of the sample data values to the point, such that data values close to p have a larger effect than further away. The basic kernel estimates may be written compactly by

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} \theta \left(\frac{x - x_i}{h} \right) \tag{5.11}$$

Where represents each data point in the sample of size n, and the function is a standard normal distribution with mean 0 and variance 1.

$$\theta(x) = \frac{1}{\sqrt{2\pi}} \exp \left(-\frac{1}{2} x^2 \right) \tag{5.12}$$

where $\theta(x) = \frac{1}{h} \theta \left(\frac{x}{h} \right)$ (notation introduced by [83]). The kernel estimate can be motivated not only as the limiting case of the averaged shifted histogram (ASH). The smoothness of the estimate depends on the parameter h, known as the bandwidth. If h is small, only data values vary close to the point p have influence on the estimated density, and this tends to make the estimate rather

jagged. As h increases, data values further away from p start to influence the distribution, and it tends to became smoother.

5.2.3. Gaussian Mixture Models

Mixture Models are a type of density model which comprise a number of component functions, usually Gaussian. These component function are combined to provide a multimodal density. Mixture models are a semi-parametric alternative to non-parametric histograms [80] (which can also be used as densities) and provide greater flexibility and precision in modeling the underlying statistic of sample data. Hopefully, the reader can see the connection between finite mixtures and kernel density estimation. Recall that in the case of univariate kernel density estimators, we obtain these by evaluating a weighted kernel centered at each sample point, and adding these n terms. So, a kernel estimate can be considered a special case of a finite mixture where $c = n$. Therefore, the estimate of a finite mixture would be written as

$$\hat{f}_{FM}(x) = \sum_{i=1}^{c} \hat{p}_i \theta\left(x; \hat{\mu}_i, \hat{\sigma}_i^2\right) \tag{5.13}$$

where $\theta\left(x; \hat{\mu}_i, \hat{\sigma}_i^2\right)$ denotes the normal probability density function with mean μ_i, variance σ_i^2, and FM is a finite mixture.

5.2.4. Designing Cauchy RBF Neural

Figure 2.2 depicts the architecture for a fully connected RBF network. The network consists of n input features x, M hidden units with center C_j and y output. The θ_j are the basis functions, and w_{kj} are the output layer weights. The basis function activations are then calculated using a method which depends on the nature of the function. Suppose at a set of fixed point $x_1, \ldots, x_j$, $\theta_j = \theta_j(x)$ can be as in equations 2.12, 2.13, 2.14 and 2.15. We shall write the RBF network mapping in the form stated by 2.11.

5.2.5. Proposed reduction RBF to identification

Consider using a radial basis function (RBF) network to approximate a known density estimates. One hundred training data were generated from mixture Gaussian,

$$z_x = 1{,}0\theta(x; 0, 1) + 0{,}2\theta(x; 0, 1) \tag{5.14}$$

where the input x was normally distributed in $[0, 1]$. There are 1000 testing data (x_1, z_i) with randomly distributed in the range $(0, 1)$. Here we generate 1000 data sets, independently from z. The data set are indexed by $l = 1, \ldots, L$, where $L = 100$, and for each data set we fit a model with 100 Gaussian or Cauchy or Multiquadric or Multiquadric Inverse and regularized by $\lambda = 0{,}001$ to give an approximate prediction function.

The Gaussian basis function was used with a kernel $\sigma = 0{,}1$. All the 100 training data points were used as the candidate RBF center set for centre c.

The design matrix from the input data, centre positions and radial factors has size of 100×100. We assume that $w = \text{inv}(\theta^T\theta)\theta^T y$ and $\phi = \theta w$ with 100 neurons has been obtained. The network output for an input x_i is given by 2.11.

The target function to be approximately is the following one density estimate function. Training samples (x, y) estimate density kernel. The number of training samples kernel Box, Epanechnikov, and Triangle are 100. The approximation accuracy is estimated for test samples after incremental learning is completed. The test samples are also randomly drawn from the same regions, and the numbers of them are 1000 to Box, Epanechnikov, and Triangle, respectively. The estimate is based on a RBF reduced by QLP decomposition. The density is evaluated at 100 equally-spaced points covering the range of the data in x.

Each iteration with a RBF network requires a single matrix inversion. The first process is the creation of the matrix design ϕ composed with inputs and centre. Here the matrix design has same radios and centers equal the data input.

5.2.6. RBF type Inverse Multiquadric to Kernel Density Estimation

The QLP decomposition in the figure 5.5 for the RBF reduced by QLP, the pruning threshold is chosen as 10 neurons. Initial the 100 training data points were used to model as the candidate RBF centre set and the regularization parameter was fixed to. One method to choose this number of hidden is to use the minimum value of the BIC criterion.

Table 5.3, shows us the error squared mean with the different kernel density estimate about the training set and testing set. A good resulted is obtained with Triangle density estimate. If consider 10 neurons the MSE would be $9{,}64 \times 10^{-6}$ for training and 0,000150 for testing.

Comparison of BIC to the case density estimate normal the values of 10 neurons against 7, 9 and 12 neurons were made. Table 5.4, shows us the least

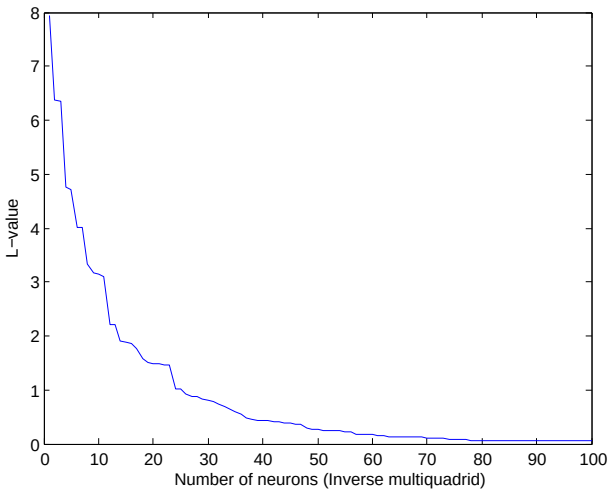


Figura 5.5: QLP Decomposition in the Inverse Multiquadric case

BIC value of $1{,}04 \times 10^{-6}$ without pruning and a value 0,00010 with pruning by QLP decomposition in the matrix design.

Figure 5.6, 5.7, 5.8 and 5.9 shows that RBF reduced by QLP is better in Triangle, Epanechnikov and Normal in comparison with Box desnity. In table 5.3, the inverse multiquadratic RBF with kernel triangle case presents a minor error squared mean (MSE) to training and test in comparison to other cases. The final prediction error value with QLP was $3{,}59 \times 10^{-5}$ and final prediction error was of $2{,}19 \times 10^{-5}$.

Density	Error training	Error test
Box	$4{,}85 \times 10^{-5}$	0,000165
Triangle	$9{,}64 \times 10^{-6}$	0,000150
Epanechnikov	$2{,}80 \times 10^{-5}$	0,000159
Normal	$3{,}39 \times 10^{-5}$	0,000512

Cuadro 5.3: Error squared mean of method using RBF reduced by QLP decomposition (Cauchy).

EPF=error prediction final and BIC=Schwartz Bayesian Information Criterion

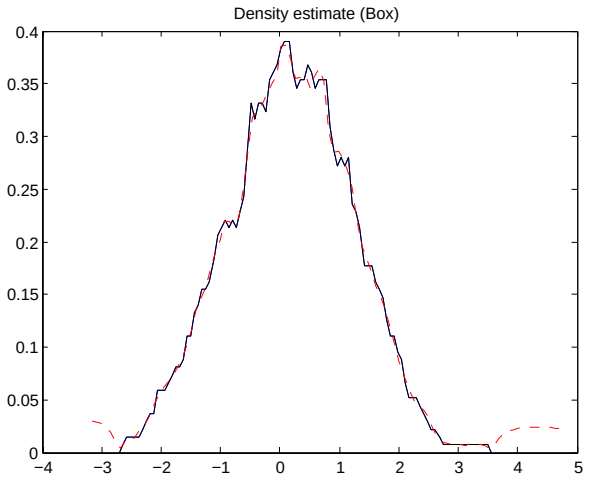


Figura 5.6: Kernel Density Estimation (black solid line) and Inverse Multiquadric RBF reduced by QLP (red solid line)

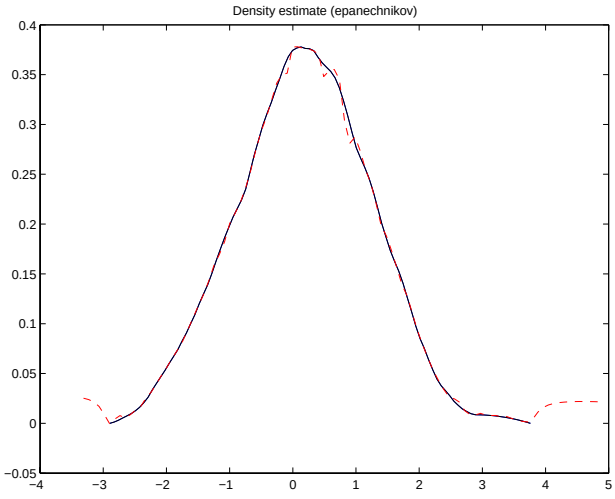


Figura 5.7: Kernel Density Estimation (black solid line) and Inverse Multiquadric RBF reduced by QLP (red solid line)

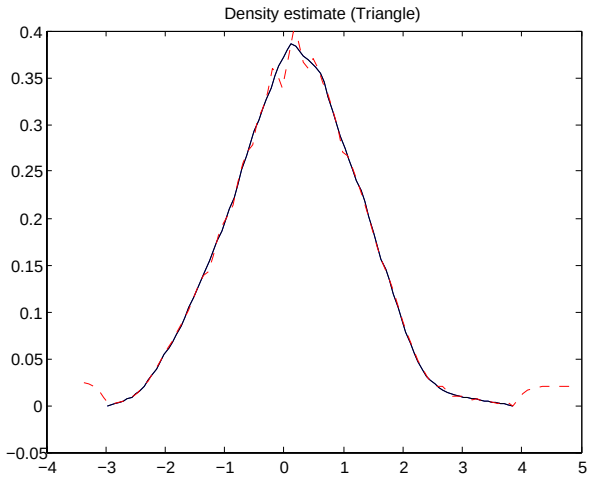


Figura 5.8: Kernel Density Estimation (black solid line) and Inverse Multiquadric RBF reduced by QLP (red solid line)

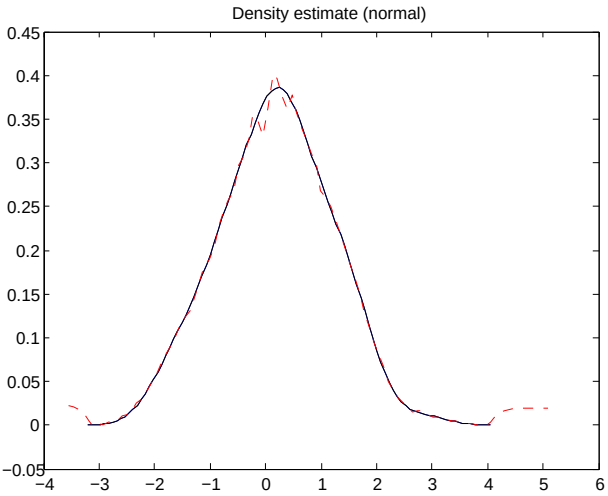


Figura 5.9: Kernel Density Estimation (black solid line) and Inverse Multiquadric RBF reduced by QLP (red solid line)

	BIC		EPF	
	Φ	Φ_{QLP}	Φ	Φ_{QLP}
Box	$7{,}85 \times 10^{-5}$	$18{,}1 \times 10^{-5}$	$4{,}41 \times 10^{-5}$	$11{,}1 \times 10^{-5}$
Triangle	$1{,}73 \times 10^{-6}$	$3{,}59 \times 10^{-5}$	$9{,}63 \times 10^{-7}$	$2{,}19 \times 10^{-5}$
Epanechnikov	$1{,}04 \times 10^{-6}$	0,00010	$1{,}04 \times 10^{-6}$	$6{,}30 \times 10^{-5}$
Normal	$1{,}05 \times 10^{-6}$	0,00012	$5{,}63 \times 10^{-7}$	$7{,}37 \times 10^{-5}$

Cuadro 5.4: Numerical results for BIC and EPF after reduction of matrix design

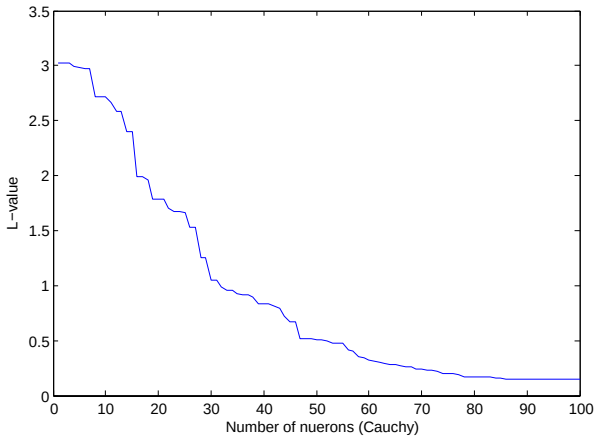


Figura 5.10: Decomposition QLP in the case Cauchy

5.2.7. RBF type Cauchy to Kernel Density Estimation

The QLP decomposition for the RBF reduced by QLP in figures 5.11, 5.12, 5.13 and 5.14, show that the pruning threshold was chosen as 40 neurons.

Comparison of BIC to the Normal case also the values of 40 against 30, 35 and 50 neurons were made. In this case (Cauchy) shows that the re-estimative to all kernel density by RBF reduced by QLP decomposition was less successful. The density estimates are roughly comparable, but the normal kernel produces a density that is rougher than the others.

5.2.8. RBF type Multiquadric to Kernel Density Estimation

The QLP decomposition in the 5.15 for the RBF reduced by QLP, the pruning threshold is chosen as 12 neurons also.

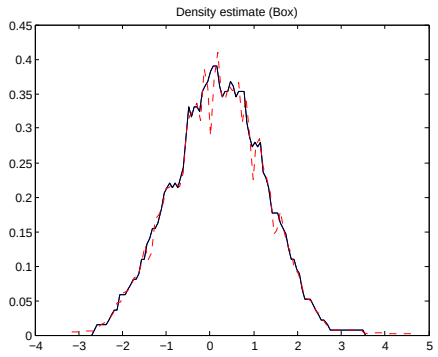


Figura 5.11: Kernel Density Estimation (black solid line) and Cauchy RBF reduced by QLP (red solid line) Box

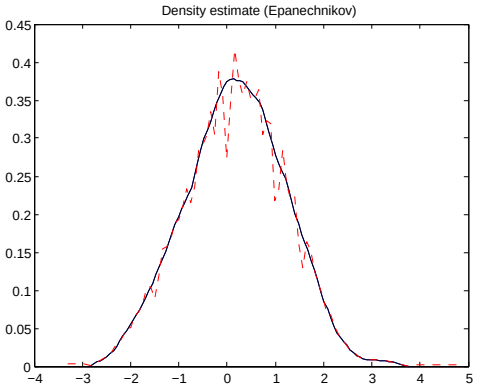


Figura 5.12: Kernel Density Estimation (black solid line) and Cauchy RBF reduced by QLP (red solid line) Epanechnikov

Density	Error training	Error test
Box	0,00029	0,00016
Triangle	0,00033	0,00015
Epanechnikov	0,00038	0,00010
Normal	0,00057	0,00010

Cuadro 5.5: The error squared mean of method using RBF reduced by QLP decomposition (Cauchy)

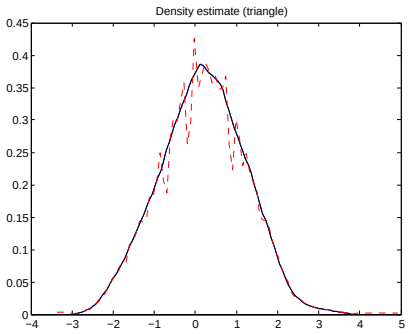


Figura 5.13: Kernel Density Estimation (black solid line) and Cauchy RBF reduced by QLP (red solid line) Triangle

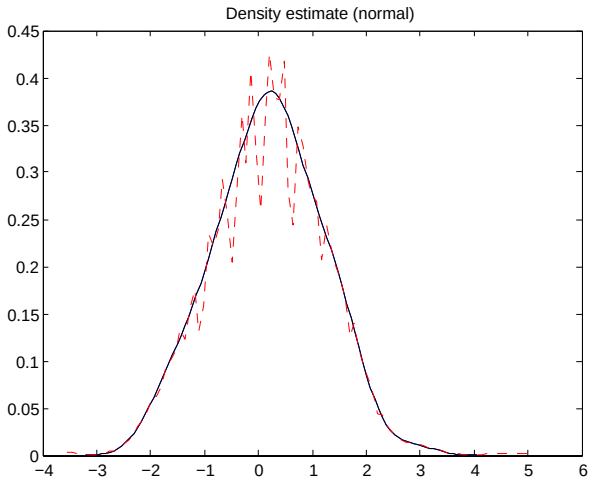


Figura 5.14: Kernel Density Estimation (black solid line) and Cauchy RBF reduced by QLP (red solid line) Normal

	BIC		FPE	
	Φ	Φ_{QLP}	Φ	Φ_{QLP}
Box	$7{,}37 \times 10^{-5}$	0,00110	$3{,}82 \times 10^{-5}$	0,00065
Triangle	$8{,}40 \times 10^{-6}$	0,00073	$4{,}28 \times 10^{-6}$	0,00073
Epanechnikov	$7{,}63 \times 10^{-6}$	0,00144	$3{,}91 \times 10^{-6}$	0,00085
Normal	$1{,}03 \times 10^{-5}$	0,00216	$5{,}20 \times 10^{-6}$	0,00127

Cuadro 5.6: Numerical results for BIC and EPF after reduction of matrix design

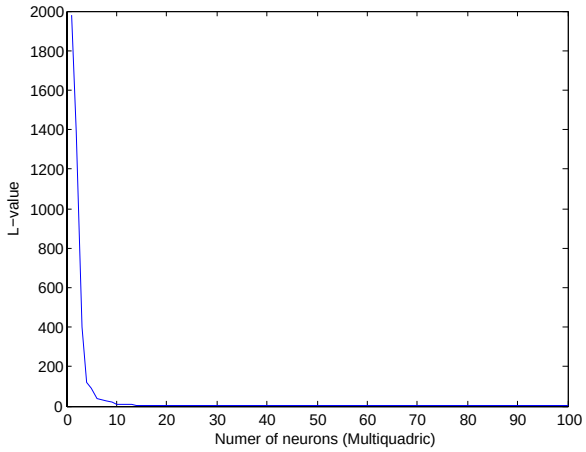


Figura 5.15: QLP decomposition in the Multiquadric case

Density	Error training	Error test
Box	0,00031	$1{,}40 \times 10^{-4}$
Triangle	$4{,}06 \times 10^{-5}$	0,00014
Epanechnikov	$3{,}67 \times 10^{-5}$	$1{,}39 \times 10^{-4}$
Normal	$9{,}09 \times 10^{-6}$	0,00014

Cuadro 5.7: Error squared mean of method using RBF reduced by QLP decomposition (Multiquadric)

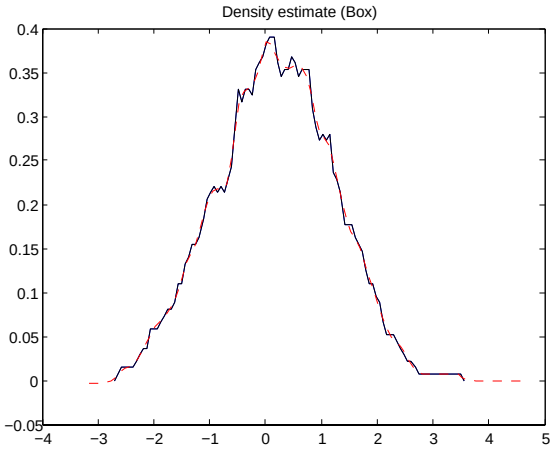


Figura 5.16: Kernel Density Estimation (black solid line) and Multiquadric RBF reduced by QLP (red solid line) Box

Figures 5.16, 5.17, 5.18 and 5.19 consist of a linear combination of 12 radial functions to re-estimate all kernel density.

In this case kernel density by RBF reduced by QLP decomposition was less successful in their tail behavior (Figure 8). In this case the area of probability density estimate is not 1. Comparison of BIC values of 12 against 10, 15 and 20 neurons were made, the value was a value of $8{,}59 \times 10^5$ to BIC without reduction QLP and $5{,}96 \times 10^5$ to matrix design with reduction QLP.

	BIC		FPE	
	Φ	Φ_{QLP}	Φ	Φ_{QLP}
Box	$8{,}44 \times 10^{-5}$	0,00024	$4{,}96 \times 10^{-5}$	0,00024
Triangle	$7{,}69 \times 10^{-7}$	$6{,}60 \times 10^{-5}$	$4{,}44 \times 10^{-7}$	$5{,}16 \times 10^{-5}$
Epanechnikov	$8{,}59 \times 10^{-5}$	$5{,}96 \times 10^{-5}$	$5{,}96 \times 10^{-7}$	$4{,}66 \times 10^{-5}$
Normal	$7{,}00 \times 10^{-10}$	$1{,}47 \times 10^{-5}$	$3{,}99 \times 10^{-10}$	$1{,}15 \times 10^{-5}$

Cuadro 5.8: Numerical results for BIC and EPF after reduction of matrix design

5.2.9. Conclusion

In this paper, a simple idea of using RBF reduction by QLP decomposition to approximate density estimate has been developed. The experimental

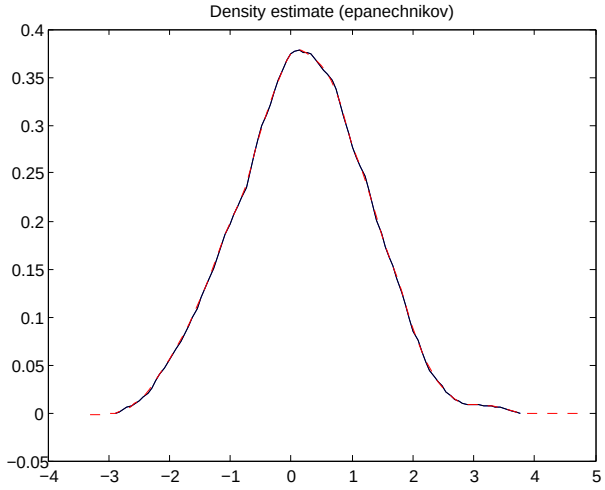


Figura 5.17: Kernel Density Estimation (black solid line) and Multiquadric RBF reduced by QLP (red solid line) Epanechnikov

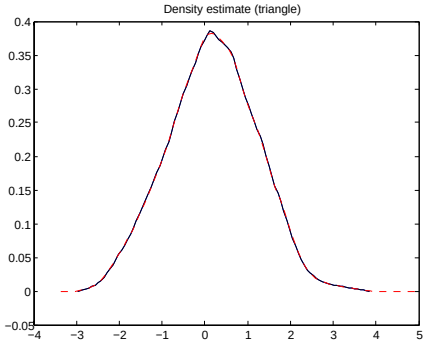


Figura 5.18: Kernel Density Estimation (black solid line) and Multiquadric RBF reduced by QLP (red solid line) Triangle

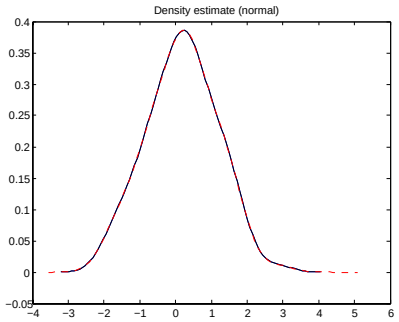


Figura 5.19: Kernel Density Estimation (black solid line) and Multiquadric RBF reduced by QLP (red solid line) Normal

results demonstrate the potential of our proposed techniques, indicating that QLP is effective when the RBF centre aren't adjusted and the regularization parameters are kept fixed. The value BIC to minor number of neurons confirm the QLP decomposition. Figures 5.16, 5.17, 5.18 and 5.19, show that the RBF network estimate after the reduction in the number of hidden units, a good result is obtained with 10 neurons in the case kernel density estimate Triangle, Epanechnikov and Normal cases with function RBF type inverse multiquadric. We also showed that mean square error of selection RBF for training and testing has a better value in the case Inverse multiquadric. The re-estimative of the Cauchy and inverse multiquadric RBF type function were less successful for all kernel estimates. We conclude that specific density estimate can be accurately carried out using a multiquadric RBF neural network pruning with QLP decomposition.

Apéndice A

Aspectos técnicos

A.1. Algunos resultados de Álgebra Matricial

A.1.1. El número de condición de una matriz

Sea A una matriz no singular. El número de condición de A, denotado cond(A), se define como cond$(A) = \|A\| \, \|A^+\|$. Si cond(A) es pequeño, entonces se dice que la matriz está bien condicionada. Si cond(A) es grande, entonces A está mal condicionada. Este número afecta a la manera en que se comporta la matriz en cuanto a la resolución de sistemas de ecuaciones. Ahora se verá cómo se puede usar cond(A) para indicar la precisión de la solución de un sistema de ecuaciones $AX = B$. Supongamos que esta ecuación describe un experimento dado y que los elementos de A y B provienen de mediciones. Estos datos dependen de la precisión de los instrumentos, y es raro que sean exactos. Se representarán los errores en A mediante una matriz E y los correspodientes errores en X por e, de manera que $(A + B)(X + e) = B$. Escogiendo normas adecuadas para los errores y las matrices, se puede demostrar que

$$\frac{\|e\|}{\|X + e\|} \leq \text{cond}(A) \frac{\|E\|}{\|A\|} \tag{A.1}$$

Por lo tanto, si cond(A) es pequeño, errores pequeños en A sólo pueden producir errores pequeños en X y el resultado será correcto. Se dice que el sistema de ecuaciones es bien condicionado. Por otro lado, si cond(A) es grande, existe la posibilidad de que errores pequeños en A produzcan errores grandes en X llevando a resultados muy inexactos. De un sistema de estos, de dice que está mal condicionado. Observe que un valor grande de cond(A) es

una advertencia, no una garantía, de un posible error grande en la solución. Es claro que el valor de cond(A) dependerá de la norma que se use para A, se usa con frecuencia se denomina la norma

$$\|A\| = \max\left(|a_{1j}| + \cdots + |a_{nj}|\right) \quad j = 1, \ldots, n \tag{A.2}$$

De esta forma, la interpretación de la condición de A es la de un factor de amplificación de los errores en los datos al intentar lograr una solución de un problema de mínimos cuadrados [48].

Los métodos y algoritmos de reducción de dimensionalidad mejoran la condición numérica de una matriz, dando lugar a representaciones matriciales más robustas para los datos de entrada e intermedios. Si rango(A) $= n$ para una matriz A de $m \times n$ donde $m > n$, cualquier matriz $m \times r, r < n$ obtenida a partir de A mediante la supresión de $n - r$ columnas (neuronas) verificará cond(B) $\leq$ cond(A) [48], donde r son las neuronas descartadas tras la reducción según el QLP. Por consiguiente, dichos métodos de reducción de dimensionalidad pueden conllevar una pérdida de información más o menos significativa, pero en ningún caso producirán un empeoramiento de los problemas numéricos.

Como conclusión la descomposición QLP resulta sumamente útil para reducción de dimensionalidad, detección de información redundante y determinación del número óptimo de variables linealmente independientes, en nuestro caso el ajuste de la función $SinE$ y, por añadidura, en los problemas de predicción de series caóticas planteados en este trabajo.

La salida de la red RBF resuelve el sistema $w2 = a1/t$, donde t es el objetivo (*target*) y $a1$ es la matriz de diseño de la red (En Matlab® , `[w2,b2]=solvelin2(a1,t)`). Muchas veces aparece un rango deficiente por causa de la no singularidad principalmente en la matriz de retardos. En Matlab® la función **cond** calcula el número de condición. Nuestro objetivo es encontrar una estimación relativa del error que ocurre al calcular la solución de la ecuación $\phi = wt$

Teorema A.1.1 *Sea ϕ una matriz inversible y consideremos cond(ϕ) para alguna norma $\|\phi\|$ que satisfaga la propiedad de consistencia. Supongamos que w es la solución para $\phi = wt$, w_c es la solución computada, y $r = \phi w_c - b$ es el residuo. Entonces,*

$$\frac{\| w_c - w_t \|}{w_t} \leq \text{cond}(\phi)\frac{r}{t} \tag{A.3}$$

A.1.2. Descomposición QLP en el entorno Matlab®

La matriz A de tamaño $m \times n$ se puede descomponer como

$$\begin{aligned} [Q, R, pr1] &= qr(A, 0) \\ (Q1, R1, pr1) &= qr(R', 0) \\ Aqlp &= Q \cdot R1' \cdot Q1 = A \\ Aqlpinv &= Q1 \cdot inv(R1') \cdot Q \end{aligned}$$

La matriz Q obtenida es ortogonal y R es una matriz triangular superior ($R(i,j) = 0$ para $i > j$ y $R1$ es triangular inferior. Para que el producto QR sea posible, Q será $m \times m$. Se verifica que si rango$(A) = n$, A es de rango completo, las primeras n columnas de Q forman una base ortogonal para el espacio engendrado por A.

A.1.3. Dependencia lineal

Dada la matriz de diseño compuesta de centros y radios, una columna es linealmente dependiente si existen los escalares $k_1, \ldots, k_n$, no todos nulos, tales que:

$$k_1 x_1 + \cdots + k_n x_n = 0 \tag{A.4}$$

Si una matriz de columnas de vectores no es linealmente dependiente, diremos que las columnas de la matriz de diseño son linealmente independientes. En el contexto de este trabajo un conjunto de vectores linealmente independientes corresponde a unas columnas que no están relacionadas linealmente de forma exacta.

A.1.4. Proyecciones y número de parámetros de la red

Se define la matriz de proyección como:

$$P = I_p - \theta A^{-1} \theta^T \tag{A.5}$$

donde, $A^{-1} = (\theta^T \theta - L)^{-1}$ está relacionada con la matriz θ, que a su vez está formada por las evaluaciones de las funciones radiales para cada nodo e I_p denota la matriz identidad de orden p. La matriz $L = U^T U$ representa la matriz que contiene el parámetro de regularización, L es una matriz diagonal que contiene los parámetros de regulación l_i utilizados para encontrar los pesos w_i de la red neuronal. El número efectivo de parámetros $\gamma = p -$ traza(P). Adicionalmente se define Q, como la suma de los cuadrados de los errores entre los valores predichos y los de comprobación.

A.2. Contraste de Kolmogorov-Smirnov

En el entorno Matlab® este contraste se realiza con la función `kstest2`. La función de distribución empírica, $F_n(x)$, de una muestra, $x_1, x_2, \ldots, x_n$, se define como el cociente entre el número de valores del conjunto $x_1, x_2, \ldots, x_n$ que son menores o iguales que x y el tamaño de la muestra, n.

Para contrastar la hipótesis de que la muestra se ajusta a una distribución teórica, $F(x)$, que en nuestro caso puede ser una Gamma, Lognormal o Weibull, se calcula el estadístico:

$$D_n = \text{máx}\,|F_n(x) - F(x)| \tag{A.6}$$

cuya distribución es conocida y está tabulada. Si la distancia calculada, D_n, es mayor que la encontrada en las tablas, para un nivel α, rechazamos la distribución $F(x)$ para la muestra. Para n y α dados, hallamos $D(\alpha, n)$ tal que $P(D_n > D(\alpha, n)) = \alpha$. La región crítica del contraste será $D_n > D(\alpha, n)$. Este contraste tiene la ventaja de que no requiere agrupar los datos y el inconveniente de que si calculamos $F(x)$ estimando parámetros de la población, mediante la muestra, la distribución de D_n es sólo aproximada.

Se puede utilizar para estos contrastes el criterio del P-valor, rechazando la hipótesis nula al nivel α cuando el P-valor es menor que α, y aceptándola en caso contrario. En el entorno Matlab® se considera el valor $h = 1$ rechazando la hipótesis nula, y $h = 0$ aceptando la misma.

A.3. Cálculos de aprendizaje

A.3.1. Función de coste

Como $P = P^T$ la función de coste 2.19 en forma matricial puede ser reescrita como:

$$\begin{aligned} \zeta_{\text{Tikhonov}} &= (\theta\hat{w} - \hat{d})^T(\theta\hat{w} - \hat{d}) + \hat{(}w)^T\lambda\hat{w} = \\ &= \hat{d}^T(\theta V^{-1}\theta^T - I_p)(\theta V^{-1}\theta - I_p)\hat{d} + \hat{d}^T\theta V^{-1}\lambda V^{-1}\theta^T\hat{d} = \\ &= \hat{d}P^T\hat{d} + \hat{d}^T\theta V^{-1}\lambda V^{-1}\theta^T\hat{d} \end{aligned} \tag{A.7}$$

Como, $\theta V^{-1}\lambda V^{-1}\theta^T = \theta V^{-1}(V - \theta^T\theta)V^{-1}\theta^T$, podemos escribir:

$$\begin{aligned} \zeta_{\text{Tikhonov}} &= \theta V^{-1}\theta^T - (\theta V^{-1}\theta^2)^2 = \\ &= P - P^2 = \hat{d}^T P\hat{d} + \hat{d}^T(P - P^2)\hat{d} = \hat{d}^T P\hat{d} \end{aligned} \tag{A.8}$$

A.3.2. Fase supervisada

Se calculan los pesos y umbrales de las neuronas de salida de la red. En este caso, el objetivo es minimizar las diferencias entre las salidas de la red y las salidas deseadas. Por tanto, el proceso de aprendizaje está guiado por la minimización de una función error computada de la red.

Mínimos cuadrados

Para resolver este problema de optimización se suele utilizar una técnica basada en la corrección del error. En la ecuación

$$\phi_k(n) = \sum_{i=1}^{m} w_{ik}\theta_i(n) + u_k \qquad k = 1, 2, \ldots, r \tag{A.9}$$

se observa que las salidas de la red de base radial dependen linealmente de los pesos y umbrales, por lo que un método bastante simple y eficiente es el algoritmo de los mínimos cuadrados. De este modo, los pesos y umbrales de la red se determinan mediante un proceso iterativo gobernado por las siguientes condiciones:

$$w_{ik} = w_{ik}(n-1) - \alpha_1 \frac{\partial e(n)}{\partial w_{ik}} \tag{A.10}$$

$$u_k(n) = w_k(n-1) - \alpha_1 \frac{\partial e(n)}{\partial u_k} \tag{A.11}$$

para $k = 1, 2, \ldots, r$ e $i = 1, 2, \ldots, m$, donde $e(n)$ es el error ya definido anteriormente:

$$e(n) = \frac{1}{2} \sum_{k=1}^{r} (d_k(n) - \phi_k(n))^2 \tag{A.12}$$

y α_1 es la razón o tasa de aprendizaje. Teniendo en cuenta la expresión del error, A.12, y que el peso, w_{ik}, y el umbral, u_k, únicamente afectan a la neurona de salida k, se obtiene que:

$$\frac{\partial e(n)}{\partial w_{ik}} = -(d_k(n) - \phi_k(n)) \frac{\partial \phi_k(n)}{\partial w_{ik}} \tag{A.13}$$

$$\frac{\partial e(n)}{\partial u_k} = -(d_k(n) - \phi_k(n)) \frac{\partial \phi_k(n)}{\partial u_k} \tag{A.14}$$

Derivando la salida $\phi_k(n)$ de la red de base radial dada en la ecuación A.9 respecto a los pesos y umbrales, se obtiene que:

$$\frac{\partial e(n)}{\partial w_{ik}} = \theta_i(n) \tag{A.15}$$

donde $\theta_i(n)$ es la activación de la neurona oculta i para el patrón de entrada $X(n)$, y

$$\frac{\partial \phi_k(n)}{\partial w_{ik}} = 1 \tag{A.16}$$

Por tanto, las leyes dadas por las ecuaciones A.10 y A.11 para adaptar los pesos y umbrales de la capa de salida de la red de base radial se pueden escribir de la siguiente forma:

$$w_{ik}(n) = w_{ik}(n-1) + \alpha_1(d_k(n) - \phi_k(n))\phi_i(n)) \tag{A.17}$$

$$u_k(n) = u_k(n-1) + \alpha_1(d_k(n) - \phi_k(n)) \tag{A.18}$$

para $k = 1, 2, \ldots, r$ y para $i = 1, \ldots, m$

Cuando se calculan los pesos mediante la ley de aprendizaje dada por las ecuaciones A.17 y A.18, la convergencia es bastante rápida, consiguiendo una solución en un conjunto pequeño de iteraciones o ciclos de aprendizaje.

A.3.3. Método de aprendizaje totalmente supervisado

Este método no conserva, en principio, las propiedades o características locales de las redes de base radial. En este caso, todos los parámetros de la red de base radial –centros, amplitudes, pesos y umbrales– se determinan de manera completamente supervisada y con el objetivo de minimizar el error cuadrático medio, A.12. De este modo, los centros, amplitudes, pesos y umbrales de la red se modifican para cada patrón, $X(n)$, de acuerdo con las leyes de aprendizaje A.10 y A.10, y:

$$c_{ij} = c_{ij}(n-1) - \alpha_2 \frac{\partial e(n)}{\partial c_{ij}} \tag{A.19}$$

$$d_i(n) = d_i(n-1) - \alpha_3 \frac{\partial e(n)}{\partial d_i} \tag{A.20}$$

para $j = 1, 2, \ldots, p$, $i = 1, 2, \ldots, m$ y $k = 1, 2, \ldots, r$, donde α_2 y α_3 son las razones o tasas de aprendizaje para los centros y amplitudes, respectivamente.

En el caso de las redes de neuronas de base radial, la utilización del método de descenso del gradiente no implica una retropropagación del error. En

el contexto de las redes de base radial, la aplicación del método de descenso del gradiente implica el cálculo de la derivada del error, $e(n)$, respecto a cada uno de los parámetros –centros, amplitudes, desviaciones, pesos y umbrales– Dichas derivadas poseen expresiones diferentes, ya que cada uno de estos parámetros intervienen de manera distinta en la salida de la red. A continuación, se desarrollan las leyes de aprendizaje para los parámetros de las redes de base radial.

Para los pesos y umbrales las derivadas del error, $e(n)$, respecto a los pesos y umbrales de la red han sido deducidas anteriormente, obteniendo las leyes de aprendizaje dadas por las ecuaciones A.17 y A.18. Para los centros, teniendo en cuenta la expresión del error, $e(n)$, y aplicando la regla de la cadena para derivar, la derivada de dicho error respecto al parámetro c_{ij} viene dada por:

$$\frac{\partial e(n)}{\partial c_{ij}} = -\sum_{k=1}^{r}(d_k(n) - \phi_k(n))\frac{\partial \phi_k(n)}{\partial c_{ij}} \tag{A.21}$$

El parámetro c_{ij} –coordenada j del centro i– sólo interviene en la activación de la neurona oculta i, por lo que para derivar la salida k de la red, A.9, sólo es necesario derivar el término i del sumatorio, resultando entonces:

$$\frac{\partial e(n)}{\partial c_{ij}} = -\sum_{k=1}^{r}(d_k(n) - \phi_k(n))w_{ik}\frac{\partial \theta_i(n)}{\partial c_{ij}} \tag{A.22}$$

Aplicando de nuevo la regla de la cadena para derivar la función ϕ_i respecto a c_{ij} se obtiene que:

$$\frac{\partial \theta_i(n)}{\partial c_{ij}} = \theta_i(n)\frac{(x_j(n) - c_{ij})}{d_i^2} \tag{A.23}$$

Sustituyendo la ecuación A.23 en A.22, la ley para modificar los centros de las funciones de base radial dada por la ecuación A.19 adopta la siguiente expresión:

$$c_{ij} = c_{ij}(n-1) - \alpha_2\left(\sum_{k=1}^{r}(d_k(n) - \phi_k(n))w_{ik}\right)\theta_i(n)\frac{(x_j(n) - c_{ij})}{d_i^2} \tag{A.24}$$

para $j = 1, 2, \ldots, p$ e $i = 1, 2, \ldots, m$. Para las amplitudes, al igual en el caso anterior, para obtener la derivada del error, $e(n)$, respecto al parámetro d_i es necesario derivar las salidas de la red respecto a dicho parámetro:

$$\frac{\partial e(n)}{\partial d_i} = -\sum_{k=1}^{r}(d_k(n) - \phi_k(n))w_{ik}\frac{\partial \theta_i(n)}{\partial d_i} \tag{A.25}$$

La derivada de la función θ_i respecto al parámetro d_i es:

$$d_i(n) = d_i(n-1) - \alpha_3 \left(\sum_{k=1}^{r}(d_k(n) - \phi_k(n))w_{ik}\right)\theta_i(n)\frac{\|X(n) - C_i\|)^2}{d_i^3} \tag{A.26}$$

para $i = 1, 2, \ldots, m$.

A.4. Filtros

Básicamente, un filtro es un sistema que, dependiendo de algunos parámetros, realiza un proceso de discriminación de una señal de entrada obteniendo variaciones en su salida. El vector X es la entrada, el vector Y es la salida filtrada, y w es el coeficiente del filtro. En la figura A.1 se tiene un esquema del filtro utilizado en este trabajo.

$$\begin{aligned} y(n) &= b(1)x(n) + b(2)x(n-1) + \cdots + b(n+1)x(n-nb) - \\ &\quad -a(2)y(n-1) - \cdots - a(na+1)y(n-na) \end{aligned} \tag{A.27}$$

donde $n-1$ es el orden del filtro, na es el *feedback* del filtro, y nb es el *feedforward* del filtro. La salida se expresa como la convolución de la señal de entrada, $X(n)$, con las respuestas, $z(m)$.

La operación del filtro en la muestra m viene dada en el dominio del tiempo por las siguientes ecuaciones:

$$\begin{aligned} y(m) &= b(1)x(m) + z_1(m-1) \\ z_1(m) &= b(2)x(m) + z_2(m-1) - a(2)y(m) \\ &\vdots \quad \vdots \quad \vdots \\ z_{n-2}(m) &= b(m-1)x(m) + z_{n-1}(m-1) - a(n-1)y(m) \\ z_{n-1}(m) &= b(m)x(m) - a(n)y(m) \end{aligned} \tag{A.28}$$

La salida de la operación del filtro en el ámbito de transformación, Z, es una función de transferencia racional

$$y(z) = \frac{b(1) + b(2)z^{-1} + \cdots + b(bn+1)z^{-nb}}{1 + a(2)z^{-1} + \cdots + a(na+1)z^{-na}}x(z) \tag{A.29}$$

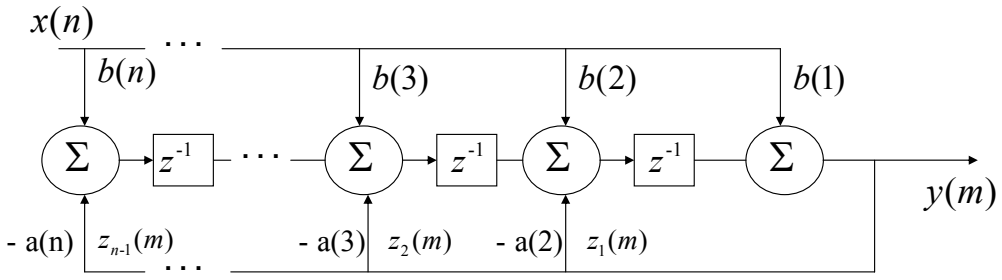


Figura A.1: Esquema del filtro (Fuente: Entorno Matlab®)

Apéndice B

Programas Matlab®

A continuación incluimos los programas que hemos desarrollado en el entorno Matlab® para esta tesis.

B.1. Filtro de Wiener: Usando la descomposición QLP, QR y SVD

```
clc;
close all;
clear all;

%Creación de la señal de entrada

var_ruido=.5;
dev_ruido=sqrt(var_ruido);
H=500;
for ii=1:1:H;
L=1500;
A=[.8 .5 -.1 -.30];
entrada=zeros(1,L);
for i=5:L
xx=0;
for j=1:4;
xi_1=xx+ entrada(i-j)*A(5-j);
end
ruido(i)=dev_ruido*randn;
objetivo(i)=xi_1;
entrada(i)=xi_1+ruido(i);
end
```

```
23 k1=1;
   while k1<11;
25
   t=cputime;
27 N=2^k1;

29 X= entrada(1,[5:N+4]);
   u= objetivo(1,[5:N+4]);
31 V=ruido(1,[5:N+4]);  % 128x128
   R1=xcorr(X)/N;  % 1
33 p1=xcorr(X,u)/N;  %

35 R2=R1(1,[N:2*N-1]);  % 1x1024
   p2=p1(1,[N:2*N-1]);  % 1024x1
37 p2=p2';
   k2=1;
39 while k2<8

41 M=2^k2;
   order=min(M,N);
43 R3=R2(1,[1:order]);
   R=toeplitz(R3);
45 p=p2([1:order],1);
   [P3,Q3,L3,pl,pr]=qlp(R);
47 Rinv=P3*inv(L3)*Q3';
   w1=Rinv*p;
49
   % Usando QR
51
   %[Qa,Ra,Ea]=qr(R);
53 %Rinv=Ea*inv(Ra)*Qa';
   %w1=Rinv*p;  % 128x1
55
   % Usando SVD
57
   %[U,S,V1]=svd(R);  %
59 %Rinv=V1*inv(S)*U';
   %w1=Rinv*p;
61
   if order < M
63 w=zeros(1,M);
   w(1,[1:order])=w1';
65 else
```

```
  w=w1([1:M],1);
67 w=w';

69 %Filtro

71 end
  Ys=filter(w,1,X);
73 Yn=filter(w,1,V);

75 Signal_power=0;
  ruido_power=0;
77 for i=1:1:length(Ys)
  Signal_power=Signal_power+Ys(i)*conj(Ys(i));
79 end
  Signal_power=Signal_power/length(Ys);
81 for i=1:1:length(Yn);
  ruido_power=ruido_power+Yn(i)*conj(Yn(i));
83 end
  ruido_power=ruido_power/length(Yn);
85
  %Razón SNR = entrada/ruido
87
  SNR1(ii,k1,k2)=(Signal_power/ruido_power);
89 T_cpu1(ii,k1,k2)=cputime-t;
  k2=k2+1;
91 end
  k1=k1+1;
93 end
  end
95
  %%%%
97
  k1=k1-1;
99 k2=k2-1;
  for i=1:1:k1;
101 for j=1:1:k2;
  SNR(i,j)=mean(SNR1(:,i,j));
103 T_cpu(i,j)=mean(T_cpu1(:,i,j));
  end
105 end
  figure(1);
107 plot([1:k1],SNR(:,2),'b.-',[1:k1],SNR(:,3),'go-',
      [1:k1],SNR(:,4),'rx:',[1:k1],SNR(:,5),'k^-',
```

```
109       [1:k1],SNR(:,6),'m>:',[1:k1],SNR(:,7),'yd:');
  grid on;
111 legend('M=4','M=8','M=16','M=32','M=64','M=128');
  ylabel('SNR')
113 xlabel('log2(tamaño de la entrada)')
  figure(2);
115 plot([1:k2],SNR(2,:),'b>-',[1:k2],SNR(3,:),'go-',
          [1:k2],SNR(4,:),'cx-',[1:k2],SNR(5,:),'m+-',[1:k2],
117       SNR(6,:),'rs-',[1:k2],SNR(7,:),'yd-',[1:k2],
          SNR(8,:),'kv-');
119 grid on;
  legend('N=4','N=8','N=16','N=32','N=64','N=128','N=256');
121 ylabel('SNR')
  xlabel('log2(orden del filtro)')
123 figure(3);
  plot([1:k1],T_cpu([1:k1],1),'r.-',[1:k1],T_cpu([1:k1],2),
125       'b.-',[1:k1],T_cpu([1:k1],3),'go-',[1:k1],
          T_cpu([1:k1],4),'rx:',[1:k1],T_cpu([1:k1],5),
127       'k^-',[1:k1],    T_cpu([1:k1],6),'y>-');
  grid on;
129 legend('M=2','M=4','M=8','M=16','M=32','M=64');
  ylabel('CPU-time')
131 xlabel('log2(tamaño de la entrada)')
  figure(4);
133 plot([1:k2],T_cpu(2,:),'b>-',[1:k2],T_cpu(3,:),
          'go-',[1:k2],T_cpu(4,:),'cx-',[1:k2],T_cpu(5,:),
135       'm+-',[1:k2],T_cpu(6,:),'rs-',[1:k2],T_cpu(7,:),
          'yd-',[1:k2],T_cpu(8,:),'kv-');
137 grid on;
  legend('N=4','N=8','N=16','N=32','N=64','N=128','N=256');
139 ylabel('CPU-time')
  xlabel('log2(orden del filtro)')
```

B.2. Probabilidades: Distribuciones $F(10,9) = F1$ y $F(9,8) = F2$

Este algoritmo también sirve para Gamma, Weibull y Lognormal

```
  clear all;
2 clc;
  fprintf(1,'EXPERIMENTO DE CONTRASTE DE
4       HIPÓTESIS CON RED RBF\n');
  fprintf(1,'--------------------------
```

```
6         ----------------------\n');
  fprintf(1,'YA!');pause
8 fprintf (1,'Generando 500 muestras de tamaño
          200 de una F(10,9)...');
10 F1=[];
  for i=1:500
12     F1=[F1 frnd(10,9,200,1)];
  end
14 save F1 F1
  fprintf (1,'ok\n') ;
16 fprintf (1,'Generando 500 muestras de tamaño 200
          de una F(9,8)...');
18 F2=[];
  for i=1:500
20     F2=[F2 frnd(9,8,200,1)];
  end
22 save F2 F2

24 a=0.1;,b=3;
  x=a+(b-a)*rand(200,1);
26 x=sort(x);
  aa=fpdf(x,10,9);
28 bb=fpdf(x,9,8);
  hold on; plot(x,aa,'b'), plot(x,bb,'k')
30


32 fprintf (1,'ok\n');
  fprintf (1,'Listo para CREAR y ENTRENAR
34          una nueva red neuronal\n');
  fprintf(1,'YA!');pause
36 genPTQLPF12;
  fprintf (1,'Red entrenada, total neuronas=
38          %d\n', net.layers{1}.size);
  fprintf (1,'Salvando red a disco...');
40 save NET7ej3 net;
  fprintf (1,'ok\n');
42 fprintf(1,'YA!');pause
  fprintf (1,'Generando para test...');
44 fprintf (1,'ok\n');
  fprintf (1,'Generando las 8 características
46          descriptivas de estas 2 muestras\n');
  fprintf(1,'YA!');pause
48 car_x=[mean(x) median(x) prctile(x,.25)
```

```
        prctile(x,.75) iqr(x)
50      sqrt(var(x)) kurtosis(x) skewness(x)]';
car_y=[mean(y) median(y) prctile(y,.25)
52      prctile(y,.75) iqr(y)
        sqrt(var(y)) kurtosis(y) skewness(y)]';
54 disp(car_x);
disp(car_y);
56 fprintf (1,'Respuesta de la red ante
        la muestra F1..');
58 resp1=sim(net,car_x);
if (resp1==1) fprintf (1,'1 (F1)\n'); end
60 if (resp1==0) fprintf (1,'0 (F2)\n'); end
fprintf (1,'Respuesta de la red ante la muestra F2...');
62 resp2=sim(net,car_y);
if (resp2==1) fprintf (1,'1 (F1)\n'); end
64 if (resp2==0) fprintf (1,'0 (F2)\n'); end
fprintf (1,'Averiguando relevancia de las 8
66      entradas originales (método QLP)...');
plot(log(abs(diag(L))))
68 fprintf (1,'ok\n');
num_ent=input('Introduzca número de entradas
70      (más relevantes) a conservar: ');
fprintf (1,'Reduciendo a las %d entradas
72      más importantes...',num_ent);
[orden,net_reduc]=red_ent(net,pr,num_ent);
74 fprintf (1,'ok\n');
fprintf (1,'Salvando red reducida a disco...');
76 save NET4ej2 net_reduc;
fprintf (1,'ok\n');
78 fprintf (1,'Respuesta de la red reducida (%d ents.)
        ante la muestraF1...',num_ent);
80 resp1=sim(net_reduc,car_x(orden));
if (resp1==1) fprintf (1,'1 (F1)\n'); end
82 if (resp1==0) fprintf (1,'0 (F2)\n'); end
fprintf (1,'Respuesta de la red reducida (%d ents.)
84      ante la muestra weibull..',num_ent);
resp2=sim(net_reduc,car_y(orden));
86 if (resp2==1) fprintf (1,'1 (F1)\n'); end
if (resp2==0) fprintf (1,'0 (F2)\n'); end
88 fprintf(1,'YA!');pause
fprintf (1,'Generando 300 muestras de tamaño
90      200 (F1) para test...');
Ft1;
```

```
92 save GT7 GT7;
   fprintf (1,'ok\n');
94 fprintf (1,'Generando 300 muestras de tamaño
          200 (F2) para test...');
96 Ft2;
   save WT7 WT7;
98 fprintf (1,'ok\n');
   fprintf(1,'YA!');pause
100 fprintf (1,'Obteniendo respuestas al test para
          la red original (8 ents.)...');
102 car_GT7=[mean(GT7)' median(GT7)' prctile(GT7,.25)'
          prctile(GT7,.75)' iqr(GT7)' sqrt(var(GT7))'
104       kurtosis(GT7)' skewness(GT7)'];
   car_WT7=[mean(WT7)' median(WT7)' prctile(WT7,.25)'
106       prctile(WT7,.75)' iqr(WT7)' sqrt(var(WT7))'
          kurtosis(WT7)' skewness(WT7)'];
108 resp_GT7_net=sim(net,car_GT7');
   resp_WT7_net=sim(net,car_WT7');
110 fprintf (1,'ok\n');
   fprintf(1,'YA!');pause
112 fprintf (1,'De 300 muestras F1, la red ha detectado
          satisfactoriamente %d\n',sum(resp_GT7_net));
114 fprintf (1,'De 300 muestras F2, la red ha detectado
          satisfactoriamente %d\n',300-sum(resp_WT7_net));
116 fprintf(1,'YA');pause
   fprintf (1,'Obteniendo respuestas al test para la
118       red reducida (4 ents.)...');
   car_GT7=[mean(GT7)' median(GT7)' prctile(GT7,.25)'
120       prctile(GT7,.75)' iqr(GT7)' sqrt(var(GT7))'
          kurtosis(GT7)' skewness(GT7)'];
122 car_WT7=[mean(WT7)' median(WT7)' prctile(WT7,.25)'
          prctile(WT7,.75)' iqr(WT7)' sqrt(var(WT7))'
124       kurtosis(WT7)' skewness(WT7)'];
   resp_GT7_net_reduc=sim(net_reduc,red_MM(car_GT7,orden)');
126 resp_WT7_net_reduc=sim(net_reduc,red_MM(car_WT7,orden)');
   fprintf (1,'ok\n');
128 fprintf(1,'YA!');pause
   fprintf (1,'De 300 muestras F1, la red ha detectado
130       satisfactoriamente %d\n',sum(resp_GT7_net_reduc));
   fprintf (1,'De 300 muestras F2, la red ha detectado
132       satisfactoriamente %d\n',
          300-sum(resp_WT7_net_reduc));
134 fprintf (1,'Preparado para reducir NEURONAS en la
```

```
        red de 4 entradas...\n');
fprintf(1,'YA!');pause
fprintf (1,'Averiguando relevancia de las %d neuronas
        originales (método QLP)...',
        net_reduc.layers{1}.size);
P_reduc=red_MM(P,orden);
A_activ=set_matriz_rbf(net_reduc,P_reduc);
[salP,salQ,salL,pl,pr_neu]=qlp(A_activ);
fprintf (1,'ok\n');
plot(log(abs(diag(salL))))
num_neu=input('Introduzca número de neuronas
        (más relevantes) a conservar: ');
fprintf (1,'Reduciendo a las %d neuronas
        más importantes...',num_neu);
[orden_neu,net_reduc2]=red_neu(net_reduc,
        pr_neu,num_neu,P_reduc,T');
fprintf (1,'ok\n');
fprintf ('Indices de las neuronas importantes...\n');
disp(orden_neu);
fprintf(1,'YA!');pause
fprintf (1,'La red reducida tiene %d entradas y
        sólo %d neuronas\n',net_reduc2.inputs{1}.size,
        net_reduc2.layers{1}.size);
fprintf (1,'Preparado para evaluar rendimiento
        de la red reducida en conjunto de test...\n');
fprintf(1,'YA!');pause
resp_GT7_net_reduc2=sim(net_reduc2,red_MM(car_GT7,orden)');
resp_WT7_net_reduc2=sim(net_reduc2,red_MM(car_WT7,orden)');
fprintf (1,'De 300 muestras F1, la red ha detectado
        satisfactoriamente %d\n',sum(resp_GT7_net_reduc2));
fprintf (1,'De 300 muestras F2, la red ha detectado
        satisfactoriamente %d\n',
        300-sum(resp_WT7_net_reduc2));
fprintf (1,'\nFin del experimento.\n');
```

B.2.1. Generar la muestra F1

```
F1=[];
for i=1:500
    F1=[F1 frnd(10,9,200,1)];
end
save F1 F1
```

B.2.2. Generar la muestra F2

```
F2=[];
for i=1:500
    F2=[F2 frnd(9,8,200,1)];
end
save F2 F2
```

B.2.3. Generar $F(10,9)$ para test

```
Ft1=[];
for i=1:300
    Ft1=[Ft1 frnd(10,9,200,1)];
end
save Ft1 Ft1
```

B.2.4. Generar $F(9,8)$ para test

```
Ft1=[];
for i=1:300
    Ft1=[Ft1 frnd(9,8,200,1)];
end
save Ft1 Ft1
```

B.2.5. genPTQLPF12.m

```
explo=randperm(1000);
P=[]; T=[];
for i=1:1000
    actual=explo(i);          %
    if (actual>500)           %
        actual=actual-500;
        col_sel=F1(:,actual);
        T(i)=1;   % resp. deseada = F1
    else
        col_sel=F2(:,actual);   %
        T(i)=0;   % resp. deseada = F2

    caract(1)=mean(col_sel);
    caract(2)=median(col_sel);
    caract(3)=prctile(col_sel,.25);
    caract(4)=prctile(col_sel,.75);
    caract(5)=iqr(col_sel);
    caract(6)=sqrt(var(col_sel));
```

```
19      caract(7)=kurtosis(col_sel);
        caract(8)=skewness(col_sel);
21      P=[P; caract];
    end
23  [P1,Q,L,pl,pr]=qlp(P);    %Ejecutar la descomposición QLP.
    pr %
25  net=newrb(P',T,1e-2);
```

B.2.6. red_ent.m

```
1   function [orden,net]=red_ent(net,pr,dim)
    orden=1:dim;
3   C=net.IW{1,1};
    libre=ones(dim,1); %
5   for i=1:dim
        if(pr(i)<=dim)
7           libres(pr(i))=0;
        end
9   end
    for i=1:dim
11      if(pr(i)>dim)
            f=find(libre);
13          C(:,f(1))=C(:,pr(i));
            libre(f(1))=0;
15          orden(f(1))=pr(i);
        end
17  end
    net.IW{1,1}=C;
19  net.inputs{1}.size=dim;
```

B.2.7. set_matriz_rbf.m

```
1   function sal=set_matriz_rbf(net,P)
    d=size(P,1);
3   H=[];
        r=net.b{1};
5       C=net.IW{1,1}; %matriz de centros (Matlab)
        for i=1:d
7           rtemp=[];
            for j=1:net.layers{1}.size
9               rtemp=[rtemp radbas(netprod(dist
                    (C(j,:),P(i,:)'),r(j)))];
11          end
            H=[H; rtemp];
```

```
13      end
        sal=H;
```

B.2.8. red_neu.m

```
function [orden,net,C]=red_neu(net,pr,dim,P,t)
2

4 orden=1:dim;
  C=net.IW{1,1};
6 r=1./(net.b{1}*sqrt(2));    % radio
  libre=ones(dim,1);
8
  for i=1:dim
10      if (pr(i)<=dim)
            libre(pr(i))=0;
12      end
  end
14
  for i=1:dim
16      if (pr(i)>dim)
            f=find(libre);          % find free
18          C(f(1),:)=C(pr(i),:);   %
            r(f(1))=r(pr(i));
20          libre(f(1))=0;          %
            orden(f(1))=pr(i);      %
22      end
  end
24 A=set_matriz_rbf(net,P);     %
  A_reduc=red_MM(A,orden);      %
26 w12=A_reduc\t;          % coefficient w12
  for i=dim+1:length(r)
28      w12=[w12; 0];     %
  end
30 net.IW{1,1}=C;                       %
  net.b{1}=1./(r*sqrt(2));              %
32 net.LW{2,1}=w12';          %
  net.layers{1}.size=dim;               % pruning of neurons
34 net.layers{2}.transferFcn='purelin';
  % purelin, hardimm.m (salida 0 % 1)
```

B.2.9. Generar Gamma

```
1 G=[];
```

```
for i=1:500
    G=[G gamrnd(5,1,200,1)-17];
end
save G G
```

B.2.10. Generar Gamma para validación

```
GT7=[];
for i=1:300
    GT7=[GT7 gamrnd(4,0.25,200,1)];
end
save GT7 GT7
```

B.2.11. Generar Weibull

```
W7=[];
for i=1:500
    W7=[W7 weibrnd(0.776,2.1,200,1)];
end
save W7 W7
```

B.2.12. Generar Weibull para validación

```
WT7=[];
for i=1:300
    WT7=[WT7 weibrnd(0.776,2.1,200,1)];
end
save WT7 WT7
```

B.2.13. Generar lognormal

```
L7=[];
for i=1:500
    L7=[L7 lognrnd(-0.0999,0.474,200,1)];
end
save L7 L7
```

B.2.14. Generar lognormal para validación

```
LT7=[];
for i=1:300
    LT7=[LT7 lognrnd(-0.0999,0.474,200,1)];
end
save LT7 LT7
```

B.3. Algoritmo de ajuste de la función SinE

```
% Generar la matriz de entrada
randn('state', 42);
rand('state', 42);
ndata = 100;
noise = 0.01;
x = unifrnd(0,10,1,ndata);
y = 0.8*exp(-0.2*x)'.*sin(2*pi*x)'+ noise*randn(ndata, 1);
c=x; r=0.1;
H=RBFGau(x,r);
w=H*y;

% Descomposición QLP

[P1,Q,L2,pl,pr]=qlp(H); %
pr1=pr(:,1:51);
H40=H(:,pr1);
w40=inv(H40'*H40)*H40'*y;
ftt=H40*w40;

% error de entrenamiento

%hold on; plot(x,y,'c+'), plot(x,ftt,'k.')
error_train=(ftt-y)'*(ftt-y)/100  % 0.0036

% test (VALIDACION)

pt=1000;
xt = unifrnd(0,10,1,pt);
yt = 0.8*exp(-0.2*xt)'.*sin(2*pi*xt)';
Ht=rbfGau(xt,r);
w=H*y;
ft1=Ht*w;

%QLP selección: predicción y grafico

Hqlpg=H(:,pr1);
wqlpg=inv(Hqlpg'*Hqlpg)*Hqlpg'*y;    %100x1
ftqlpg=Ht(:,pr1)*wqlpg;  %1000x1
%hold on; plot(x,y,'g*'), plot(xt,yt,'k.'),
%        plot(xt,ftqlpg,'r.')
errorgau=sum(ftqlpg-yt).^2/pt; % 0.1750
```

```
errorteste=(ftqlpg-yt)'*(ftqlpg-yt)/pt

% grafico de la gaussiana
cj=0.5; rj=0.1;
ht=exp(-(xt-cj).^2)/rj^2;
%hold on; plot(x,y,'r+'), plot(xt,ht,'g-')

% regularización H 100x100

hold off;
lambda=1e-3;
wg=inv(H'*H+lambda*eye(100))*H'*y; %100x1
ftg=Ht*wg; %1000x1
errorRgauH=sum(ftg-yt).^2/pt;
mseRgauH=(ftg-yt)'*(ftg-yt)/pt;

% regularización de H 100x40

hold off;
lambda=1e-3;
wqlpgr=inv(Hqlpg'*Hqlpg+lambda*eye(51))*Hqlpg'*y; %40x1
ftqlpgr=Ht(:,pr1)*wqlpgr; %
axis([0 10 -1 1]);
set(gca, 'XTick',[0 5 10])
set(gca, 'YTick',[-1 0 1])
hold on; plot(x,y,'c+'), plot(xt,yt,'k.'),
         plot(xt,ftqlpg,'r.')
errorRgauH40=sum(ftqlpgr-yt).^2/pt;
mseRgauH40=(ftqlpgr-yt)'*(ftqlpgr-yt)/pt;
egr = [];
egr=SumaLinea((Hqlpg'*y).^2)./
         (lambda+ProdDiag(Hqlpg',Hqlpg));
%

Unid = 1;
l=lambda; % Parámetro de regularización

[p, m] = size(H);
[p, k] = size(y);
if length(l) == 1
  L = diag(l * ones(m,1));
elseif length(l) == m
  L = diag(l);
```

```
85          % aqui L es una matriz (100x100) de
            % diagonales formadas por 0.001= lambda
87  else
      error('Error de predicción: Error en el
89          tamaño del parámetro de regularización')
    end
91
    [u1, u2] = size(Unid); %U=1
93  if u1 == 1 & u2 == 1
      UnidUnid = L;   %
95  elseif u1 == m & u2 == m   %
      UnidUnid = Unid' * L * Unid; %
97  else
      estr = sprintf('%d-by-%d', m, m);
99    error(['Error de predicción:
            Unid sería 1-by-1 or ' estr])
101 end

103 %
    HH = H' * H;
105
    Hy = H' * y;
107 A = inv(HH + UnidUnid);
    W = A * Hy;
109 P = eye(p) - H * A * H';
    Py = P * y;
111 yPy = ProductoTrazo(Py', Py);
    g = p - trace(P); % (ver apéndice)
113 e = [];

115       % PREDICCIÓN FINAL DEL ERROR

117       EPF = (p + g) / (p - g); % 6.2220

119       % calculo de valor BIC

121       BIC = (p + (log(p) - 1) * g) / (p - g);

123         % Calculo final del error, BIC y FPE

125       e_BIC = [e  BIC * yPy / p]; %
          e_EPF= [e  EPF * yPy / p]; %
127
```

```
        BIC_final=BIC*yPy/p;
129     EPF_final=EPF*yPy/p;

131     %Error de cada regresor

133     err22=SumaLinea((H'*y).^2)./
            (lambda+ProdDiag(H',H)); %100x1
135
        % Usando la descomposición  QLP
137

139    Unid = 1;
    l=lambda; %         regularización = 0.001
141
    [p, m] = size(Hqlpg); %100x  m=100
143 [p, k] = size(y); %k=1, p=100
    if length(l) == 1
145   L = diag(l * ones(m,1));
    elseif length(l) == m
147   L = diag(l);
    else
149   error(' Error predicción: Error en el tamaño
            del parámetro de regularización')
151 end

153 [u1, u2] = size(Unid); %Unid =1
    if u1 == 1 & u2 == 1
155   UnidUnid = L;   %
    elseif u1 == m & u2 == m   %
157   UnidUnid = Unid ' * L * Unid; %
    else
159   estr = sprintf('%d-by-%d', m, m);
      error(['Error predición: Unid seria 1-by-1 or ' estr])
161 end

163 % Cálculos diversos

165 HH1 = Hqlpg' * Hqlpg;
    Hy = Hqlpg' * y;
167 A = inv(HH1 + UnidUnid);
    W = A * Hy;  %
169 P = eye(p) - Hqlpg * A * Hqlpg';
    Py = P * y;  %
```

```
171 yPy = ProductoTrazo(Py', Py);
    g = p - trace(P); %
173

175 e = [];

177     % PREDICCIÓN FINAL DEL ERROR

179     EPFqlp = (p + g) / (p - g); %

181     %BIC

183     BICqlp = (p + (log(p) - 1) * g) / (p - g); %

185      %Cálculo final del BIC y errores

187     error_EPFqlp = [e  EPFqlp * yPy / p]; %
        error_BICqlp = [e  BICqlp * yPy / p]; %
189
        BICqlp_final = BICqlp * yPy / p; %
191     EPFqlp_final = EPFqlp * yPy / p;

193     %Error de  cada regresor

195    egrqlp = [];
    egrqlp= SumaLinea((Hqlpg'*y).^2)./
197         (lambda+ ProdDiag (Hqlpg',Hqlpg));
```

B.3.1. RBFGau.m (cálculo de la Gaussiana)

```
1 function H=RBFGau(x,Radio);
  c=x; Radio=0.1;
3 [n, p] = size(x);
  [n1, m] = size(c);
5 %Radio=0.1;
  [rr, rc] = size(Radio);
7 H = zeros(p, m);
  for j = 1:m
9     %calculo da diag D (formula de la  gaussiana)

11    %algoritmo do producto da diagonal (gaussiana)
      Diag = x- dupCol(c(:,j),p);
13 dp = ProdDiag (Diag',Diag) / Radio^2;
  gau = exp(-dp);                %gaussina RBF
```

```
15 H(:, j) = gau;                    % saida
   end
```

B.3.2. RBFMul.m (Calculo de la multicuadratica)

```
   function [Mult]=RBFMult(x,Radio);
2  c=x; Radio=0.1;
   [n, p] = size(x);
4  [n1, m] = size(c);
   %Radio=0.1;
6  [rr, rc] = size(Radio);
   G = zeros(p, m);
8  for j = 1:m
       Diag = x- dupCol(c(:,j),p);
10 dp = ProdDiag (Diag',Diag') / Radio^2;
   Mult = sqrt(dp+1);
12 Mult(:, j) = Mult;
   end
```

B.3.3. RBFInvMult.m (Calculo de la Inversa multicuadratica)

```
1  function [InvMult]=RBFMult(x,Radio);
   c=x; Radio=0.1;
3  [n, p] = size(x);
   [n1, m] = size(c);
5  [rr, rc] = size(Radio);
   G = zeros(p, m);
7  for j = 1:m
       %algoritmo del producto de la
9          diagonal (Inversa multicuadrática)
       Diag = x- dupCol(c(:,j),p);
11 dp = ProdDiag(Diag',Diag) / Radio^2;
   InvMult = 1 ./ sqrt(dp+1);
13 InvMult(:, j) = InvMult;
   end
```

B.3.4. RBFCau.m (Calculo de la Cauchy)

```
   function [Cau]=RBFCau(x,Radio);
2  c=x; Radio=0.1;
   [n, p] = size(x);
4  [n1, m] = size(c);
   [rr, rc] = size(Radio);
```

```
G = zeros(p, m);
for j = 1:m
    %algoritmo del producto de la
        diagonal (naturaleza Cauchy)
    Diag = x- dupCol(c(:,j),p);
dp = ProdDiag (Diag',Diag) / Radio^2;
Cau = 1 ./ (dp+1);
Cau(:, j) = Cau;
end
```

B.3.5. ProdDiag.m

```
function d = ProdDiag(X, Y)
[m,n] = size(X);
[p,q] = size(Y);
 if m ~= q | n ~= p
  error('ProdDiag: mala dimensión')
end
P = X';
P = P(:);
Q = Y(:);
Z = zeros(n,m);
Z(:) = P .* Q;
d = ZumaColun (Z)';
```

B.3.6. ZumaColun.m

```
function s = ZumaColun(X)

[m,n] = size(X);

if m > 1
    s = sum(X);
else
    s = X;
end
```

B.3.7. ProductoTrazo.m

```
function t = ProductoTrazo(X, Y)
  [m,n] = size(X);
[p,q] = size(Y);

if m ~= q | n ~= p
```

```
    error(' ProductoTrazo: mala dimención')
    return
end

t = sum(ProdDiag(X, Y));
```

B.3.8. SumaLinea.m

```
function s = SumaLinea(X)

[m,n] = size(X);

if n > 1
    s = sum(X')';
else
    s = X;
end
```

B.4. Descomposición QLP: qlp.m

```
function [P,Q,L,pl,pr]=qlp(X)
    [Q,R,pr]=qr(X,0);
    [P,L,pl]=qr(R',0);
    L=L';
```

B.4.1. Generar la serie caótica de Henón:

```
function sal = henon (a,b,n)%
        deseado de la serie resultante.
    A=[a; b];
    for i=1:n-2
        a=A(length(A));
        b=A(length(A)-1);
        A=[A; 1-1.4*a^2+0.3*b];
    end
    sal=A;
%v=henon(a,b,n).
%Xn+1=1-1.4X^{2}_{n}+0.3X_{n-1}
%Aqui X^{2}_{n}=ultimo(a), X_{n-1}=anterior(b)
```

B.4.2. pred_RBF.m (Predicción Henón)

```
function [P,T]=pred_RBF(v,ne,lagmax,ni)
train=v(1:ne);
```

```
test=v(ne+1:length(v));
A=set_matriz(train,lagmax);
[P,Q,L,pl,pr]=qlp(A);
plot(abs(diag(L)));
P=red_MM(A,pr(1:ni));
P=P(size(P,1)-1,:);
T=v(lagmax+1:ne);
```

B.4.3. set_matriz.m

```
function sal=set_matriz(serie,w)
  d=length(serie);
  temp=[];
  for j=1:d-w+1
    temp=[temp; serie(j+w-1:-1:j)'];
  end
  sal=temp;
```

Bibliografía

[1] Aarts, E., Korst, J.: *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley and Sons. Chichester, Reino Unido, 1989.

[2] Bareto, A.M.S.: *Algoritmo Genético dos Mínimos Quadrados Ortogonais para o Treinamiento de Redes RBF*. Tesis Doctoral. Universidade Federal do Rio de Janeiro, 2003.

[3] Brockwell, P.J., Davis, R.A. *Introducion to time Series and Forecasting*. Springer-Verlag, New York Inc. U.S.A., 1996.

[4] Broomhead, D.S., Lowe. *Multivariable functional interpolation and adaptive networks*. Complex Systems **2**, pp.321-355. 1988.

[5] Businger,P.A. y Golub,G.H.; *Linear least squares solutions by Householder transformations* Numerische Mathematik, **7**, pp. 269–276, 1965.

[6] Chakravarthy, S.V., Ghosh, J.: Scale-Based Clustering Using the Radial Basis Function Network. *IEEE Transactions on Neural Networks* **7**, pp. 1250–1261, 1996.

[7] Cheng, B., Titterington, D.M.: Neural Networks: A Review from a Statistical Perspective. *Statistical Science* **9**, pp. 2–244, 1994.

[8] Cohen, S., N. Intrator: *Global Optimization of RBF Networks*. `http://www.mat.univie.ac.at/ñeum/glopt/mss/CohI01.pdf`.

[9] Colinas, E., Rivas, C.: *Introducción a la inteligencia artificial*. Mérida, Venezuela, Universidad de Los Andes, 1998.

[10] Connel, E.H.: *Elements of Abstract and Linear Algebra*. Department of Mathematics (University of Miami). Coral Gables, Miami, U.S.A., 2002.

[11] Cox, E.: *The Fuzzy Systems Handbook: A Practicioner's Guide to Building, Using and Maintaining Fuzzy Systems*. Academic Press. Boston, Massachusetts, U.S.A., 1994.

[12] Croall, I.F., Mason, J.P (eds.): *Industrial Applications of Neural Networks*. Springer-Verlag. Berlin, Alemania, 1992.

[13] Darken, C.J., J. Moody: Towards Faster Stochastic Gradient Search. En: *Advances in Neural Information Processing Systems* **4**. Moody, J., S. Hanson y R. Lippmann (eds.). Morgan Kauffman. Palo Alto, California, U.S.A., 1992.

[14] De Carvalho, L.A.V.: *Datamining: A Mineraçao de Dados no Marketing, Medicina, Economia, Engenharia e Administraçao*. Editora Erica LDTA. São Paulo, Brasil, 2001.

[15] Drossu, R., Obradovic, Z. *Radip design of neural networks for time series prediction*. IEEE, 3, n.2,pp.78-89. 1996.

[16] Edwirde, L. S. , Lisboa, P.: *Comparison of Wiener Filter solution by SVD with decompositions QR and QLP*. WSEAS Conference in Corfu-Greece, AIKED, 2007,
http://worldses.org/programs/program-corfu2007.zip

[17] Edwirde, L. S. , Lisboa, P.: *Analysis of the characteristic features of the density functions for gamma, Weibull and log-normal distributions through RBF network pruning with QLP* . WSEAS Conference in Corfu-Greece, AIKED, 2007,
http://worldses.org/programs/program-corfu2007.zip

[18] Edwirde, L. S. : *Differenting features for the F distributions with different degrees of fredom through RBF network pruning with QLP*. WSEAS Conference in Istambul-Turkey, 2007,
http://worldses.org/programs/program-istambul2007.zip.

[19] Edwirde, L. S. : *Using RBF reduced by QLP decomposition for Probability Density Estimation*. WSEAS Conference in Hangzhou-China, AIKED, 2007,
http://worldses.org/programs/program-hangzhou2007.zip

[20] Edwirde, L. S. , Lisboa, P., González Carmona, A.: *Regression with Radial Basis Function artificial neural networks using QLP decomposition to prune hidden nodes with different functional form*. WSEAS

Conference in Vancouver-Canada, 2007,
http://worldses.org/programs/program-canada2007.zip

[21] Edwirde, L. S. , Lisboa, P., González Carmona, A.: *Pruning RBF networks with QLP decomposition to adjust* Conference in Vancouver-Canada, 2007,
http://worldses.org/programs/program-canada2007.zip

[22] Fahlman, S.E., Lebiere, C.: The Cascade Correlation Learning Architecture. In: Tourettzky, D.S. (ed.), *Advances in Neural Information Processing Systems* **2**. Morgan Kaufmann. San Mateo (California), U.S.A., 1990.

[23] Fix, E. and Hodges, J.L,Jr.: *Nonparametric Discrimination: Consistency properties* Report Number 4, USAF Scholl of aviation Medicine, Randolph Field, Texas. 1951.

[24] Flexer, A.: Connectionists and Statisticians, Friends or Foes? In: *Proceedings of the International Work Conference on Artificial Neural Networks (IWANN'95)*. Torremolinos (Málaga), España, 1995.

[25] Frances, P.H. *Time Series models for business and economic forecasting*. Cambridge, Reino Unido, 1998.

[26] Frean, M.: *The Upstart Algorithm: A Method for Constructing and Training Feed-Forward Neural Networks*. Edinburgh Physics Preprint 89/479. Department of Physics (Edinburgh University). Edinburgh, Reino Unido, 1989.

[27] Gallant, S.I.: Three Constructive Algorithms for Network Learning. In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Amherst (Cambridge), Reino Unido, 1986.

[28] Ghosh, J., Nag, A.: *An Overview of Radial Basis Function Networks*. Technical Report. Department of Electrical and Computer Engineering (University of Texas). Austin, Texas, U.S.A., 1999.

[29] Golub, G.H., Van Loan, C.F.: *Matrix Computations*. The John Hopkins University Press. Baltimore. Maryland, U.S.A., 2ª edición, 1993.

[30] Golub, G.H., Van Loan, C.F.: *Matrix Computations*. The John Hopkins University Press. Baltimore, Maryland, U.S.A., 3ª edición, 1996.

[31] Guang-Bin Huang, Saractchandran, P.: *An Efficient Sequential Learning Algorithm for Growing and Pruning RBF (GAP-RBF) Networks.* IEEE, pp. 2284-2291, Singapure, 2004.

[32] Hartman, E.J., Keeler, J.D.; *Layered Neural Networks with Gaussian Hidden Units as Universal Approximators.* Neural Computation **2**, pp. 210–215, 1990.

[33] Hastie, T., Tibshirani, R., Friedman, J.: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer-Verlag. New York, U.S.A., 2001.

[34] Haykin, S. *Adaptive Filter Theory*, 3ª edición. Prentice-Hall. Englewood Cliffs, New Jersey, U.S.A., 1996.

[35] Haykin, S.: *Neural Networks: A comprehensive Foundation.* Macmillan, New York, 1994 (y segunda edición, 2001).

[36] Heffeson, J.: *Linear Algebra Mathematics.* Saint Michael's College (Colchester), Vermont, U.S.A., 2001.

[37] Hénon, M. *A two-dimensional mapping with a strange extractor.* Communications in Mathematical Phisics bf 50, pp.69-77, 1976.

[38] Henriksson R.D., R.C. Merton: On the Market Timing and Investment Performance of Managed Portfolios II-Statistical Procedures for Evaluating Forecasting Skills. *Journal of Business* **54**, pp. 513–533, 1981.

[39] Hergert, F., Finnoff, W., Zimmermann, H.G.: A Comparison of Weight Elimination Methods for Reducing Complexity in Neural Networks. En: *Proceedings of the IEEE International Joint Conference on Neural Networks.* Baltimore, Maryland, U.S.A., 1992.

[40] Hornik, K., Stinchcombe, M., White, H.: Multilayer Feedforward Networks are Universal Approximators. *Neural Networks* **2**, pp. 359–366, 1989.

[41] Householder, A.S.; *Unitary triangularization of a monsymmtric matrix* Journal of the Association for Computing Machinery, **5**, pp. 339–342, 1958.

[42] Hwang, Y., Bank, S.: An Efficient Method to Construct a Radial Basis Function Neural Network Classifier. *Neural Networks* **10** (8), pp. 1495–1503, 1997.

[43] Intrator, O., Intrator, N.: Interpreting Neural Network Results: A Simulation Study. *Computational Statistics and Data Analysis* **37**, pp. 373–393, 2001.

[44] Isasi Viñuela, P; Galván León, I.M: *Redes de neuronas artificiales. Un enfoque práctico*, Pearson Prentice Hall, España, 2004.

[45] Kadirkamanathan, V., Niranjan, M., Fallside, F.. *Sequential Adaptation of Radial Basis Fuction Neural Networks* In advances in Neural Information Processing Systems 3 (R.P. Lippmann, J.E. Moody, and D.S. Touretzky,eds), pp.721-727. San Mateo, CA: Morgan Kaufmann. 1991.

[46] Karnin, E.D.: A Simple Procedure for Pruning Back-Propagation Trained Neural Networks. *IEEE Transactions on Neural Networks* **1**, pp. 239–242, 1990.

[47] Kushner, H.J.: *Stochastic Approximation and Recursive Algorithms and Applications*. Springer Verlag. New York, U.S.A., 2003.

[48] Lawson, C.L. y Hanson, R.J.; *Solving Least Squares problems* SIAM Publications. Philadelphia, USA, 1989.

[49] Lin, Chin-Teng, George Lee, C.S.: *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*. Prentice Hall PTR, Upper Saddle River, New Jersey, U.S.A., 1996.

[50] Lippmann, R.P.: *Neural Networks for Computing*. Lincoln Laboratory (Massachusetts Institute of Technology). Lexington (Massachusetts), U.S.A., 1988.

[51] López P.C.: *Matlab y sus Aplicaciones en las Ciencias y la Ingeniería* Prentice Hall. Universidad Complutense de Madrid, Madrid, España, 2002.

[52] Lowe, D., Webb, A. *Exploiting prior knowledge in network optimization: an illustration from medical prognosis*. Network, 1, pp.299–323., 1990.

[53] Lowe, D.: *Neural Networks for Pattern Recognition*. Oxford University Press Inc., New York, U.S.A., 1995

[54] Moody, J.E.: *The effective number of parameters: An analysis of generalization and regularization in nonlinear leraning systems.* In J.E

Moody, S.L. Hanson, and Lippmann, editors, Neural Information Processing Systems 4, pp. 879 - 854. Morgan Kaufmann CA, 1992.

[55] Moody, J., Darken, C.J.: Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation* **1**, pp. 281–294, 1989.

[56] Makridakis, S., Wheelwright, S. *Manual de técnicas de pronósticos* Ed. Limusa Noriega, México, 1994.

[57] Marks II, R.J.: Intelligence: Computational vs. Artificial. *IEEE Transactions on Neural Networks* **4**, pp.737–739, 1993.

[58] Martín, B., Sanz, A.: *Redes Neuronales y Sistemas Borrosos*. Editorial RA-MA. Madrid, España, 1997.

[59] Matlab Reference Manual, Mathworks, USA, 2007.

[60] Mézard, M., Nadal, J.P.: Learning in Feed-Forward Layered Networks: The Tiling Algorithm. *Journal of Physics A* **22**, pp. 2191–2203, 1989. `http://citeseer.ist.psu.edu/context/8465/0`

[61] Mhaskar, H.M.: Neural networks for optimal approximation of smooth and analytic functions. *Neural Computation* **8**, pp. 1731–1742, 1996.

[62] Michelli, C.A.: *Interpolation of scattered data: Distance matrices and conditionally positive definite functions, constructive Aproximation*. **37**, pp. 11-12, 1986.

[63] Qi, M.: Financial Applications of Artificial Neural Networks. En: G.S. Maddala y C.R. Rao (eds.), *Handbook of Statistics* **14**. Elsevier Science Publishing Company. New York, U.S.A., 1996.

[64] Nicolelis, M., Chapin, J.K.: *Control Cerebral de Robots*. Investigación y Ciencia **315**, pp. 6–15. Diciembre, 2002.

[65] Nirajan, M., Fallside, F. *Neural networks and radial basis functions in classifying static speech patterns*. Computer Speech and Language, 4, pp.275–289, 1990.

[66] Oppenhein, A.V and R.W. Schafer.; *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, pp.311-312, 1969.

[67] Orr, M.J.L.: Optimising the Widths of Radial Functions In: *Vth IEEE Brazilian Symposium on Neural Networks*. Belo Horizonte, Brasil, 1998.

[68] Orr, M.J.L.: *Introduction to Radial Basis Function Networks*. Technical Report. Centre for Cognitive Science (University of Edinburgh). Edinburgh, Reino Unido, 1996.

[69] Pérez, C. *Técnicas Estadísticas con SPSS 12. Aplicaciones al análisis de datos*. Person Educación, S.A. Madrid, España, 2005.

[70] Poggio, T., Girosi, F.: Networks for Approximation and Learning. *Proceeding of the IEEE* **78**, pp. 1481–1497, 1990.

[71] Powell, M.J.D.: Radial Basis Functions for Multivariate Interpolation: A Review. In: *Algorithms for Approximation*. J.C. Mason and M.G. Cox, (eds.). Clarendon Press. Oxford, Reino Unido, 1987.

[72] Powell, M.J.D.: *Radial Basis Function Methods for Interpolation to Functions of Many Variables*. Technical Report. Numerical Analysis Group NA2001/11. DAMTP, University of Cambridge, Reino Unido, 2001.

[73] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: *Numerical Recipes in C*, segunda edición. Cambridge University Press. Cambridge, Reino Unido, 1992.

[74] Refenes, A.N., Vithlani, S.: Constructive Learning by Specialization. In: *Proceedings of the International Conference on Neural Networks*. Helsinki, Finland, 1991.

[75] Refenes, A. (ed.): *Neural Networks in the Capital Markets*. John Wiley & Sons. New York, U.S.A., 1995.

[76] Refenes, A.N., Burgess, A.N., Bentz, Y. *Neural networks in financial enginering: a study in methodology*. IEEE, 8, n.6, 1997.

[77] Renk C, Tijms. *A First Course in Stochastic Models* . Wiley, Vrije Universiteit, Amsterdam, Netherlands, 2003.

[78] Rezende, S.O.: *Sistemas Inteligentes: Fundamentos e Aplicações*. Editora manole LTDA. Barueri, Brasil, 2003.

[79] Rojas, R.: *Neural Networks: A Systematic Introduction*. Springer-Verlag. Berlin, Alemania, 1996.

[80] Rosenblatt, M. *"Remarks on some Nonparametric Estimates of a density function"*. Ann. Math. Statistic, **27**, pp. 823–837, 1956.

[81] Salmerón, M.: *Predicción de Series Temporales con Redes Neuronales de Funciones Radiales y Técnicas de Descomposición Matricial*. Tesis Doctoral. Universidad de Granada, 2001.

[82] Sarle, W.S. *Neural Network and Statistic Models* In Proceedings of the Nineteenth Annual SAS Users Group International Conference, pp.1538-1550, Cary, NC, 1994.

[83] Scott, David W.: *On optimal and data-based histograms* **66**, pp.605–610, 1985.

[84] Sietsma, J., Dow, R.F.J.: Creating Artificial Neural networks that Generalize. In: Refenes, A. (ed.), *Neural Networks in the Capital Markets*. John Wiley and Sons. New York, U.S.A., 1991.

[85] Sprott, J.C.: *Chaos and Time-Series Analysis*. Oxford University Press. Oxford, Reino Unido, 2003.

[86] Stewart, G.W. *Matrix Algoritms: Basic Decompositions*. SIAM Publications. Philadelphia, U.S.A., 1998.

[87] Stewart, G.W. *On Inexpensive Triangular Approximation to the Singular value Decomposition*. Department of Computer Science and Institute for U.S.A, pp.1-16, 1998.

[88] Tikhonov, A.N., Arsenin, V.Y.: *Solutions of Ill-Posed Problems*. W.H. Winston. Washington DC, Washington, U.S.A., 1997.

[89] Valiant, L.: A Theory of the Learnable. *Communications of the ACM* **27**, pp. 1137–1142, 1984.

[90] Vapnik V.N., Chervonenkis, Y.: On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theoretical Probability and Its Applications* **17**, pp. 264–280, 1971.

[91] Vapnik V.N.: Minimum Variance Beamforming. In: *Adaptive Radar Detection and Estimation*. S. Haykin y A. Steinhardt (eds.). Wiley Interscience. New York, U.S.A., 1992.

[92] Vapnik V.N.: *Statistical Learning Theory*. Wiley, New York, U.S.A., 1998.

[93] Venturieri, A., Santos, J.R.: Técnicas de classificação de imagens para análise de cobertura vegetal. In: Assad, E.D. y E.E. Sano (eds.), *Sistemas de Informações Geográficas. Aplicações na Agricultura*, segunda edición. Embrapa-SPI/Embrapa-CPAC. Brasilia, Brasil, 1998.

[94] Vidyasagar, M.: *Learning and Generalization: With Applications to Neural Networks*, segunda edición. Springer Verlag. Londres, Reino Unido, 2003.

[95] White, H.: Some Asymptotic Results for Learning in Single Hidden Layer Feedforward Network Models. *Journal of the American Statistical Society* **84**, pp. 1003–1013, 1989.

[96] White, H.: Learning in Artificial Neural Networks: A Statistical Perspective. *Neural Computation* **1**, pp. 425–464, 1989.

[97] White, H.: *Artificial Neural Networks: Aproximation and Learning Theory*. Blackwell Publishers. Cambridge, Reino Unido, 1992.

[98] Widrow, B., Hopf, M.E. Jr.: Adaptive Switching Circuits. En: *IRE Western Electric Show and Convention Record*. Institute of Radio Engineers. New York, U.S.A., 1960.
`http://www-isl.stanford.edu/~widrow/`

[99] Widrow, B., Stearns, S.D.: *Adaptive Signal Processing*. Prentice-Hall. Englewood Cliffs, New Jersey, U.S.A., 1985.

[100] Yim, J. *Previsao de Serie de Tempo: Modelo ARIMA, Modelos Estruturais e Redes Neurais Artificiais*. Dissertacao(Economia). Universidade de Sao Paulo, Brasil, 2001.

Índice de cuadros

Índice de figuras

Índice general