Capítulo 6

RED DE BASE RADIAL (RBF)

Introducción

En este capítulo nos vamos a enfrentar a un nuevo tipo de redes neuronales que fueron introducidas en 1985. Desde el punto de vista de su arquitectura es una red multicapa unidireccional con aprendizaje híbrido, pues en la capa oculta se sigue un algoritmo No-Supervisado y en la de salida el aprendizaje es Supervisado.

Debido a su simplicidad y velocidad en el proceso de aprendizaje, y el alto grado de generalización, esta red ha sido utilizada en diversas aplicaciones prácticas, sobre todo en reconocimiento y clasificación de patrones. Otro campo de aplicación que ha presentado resultados promisorios es la identificación y modelado de sistemas no lineales.

Las redes de base radial conceptualmente surgen como un caso particular de una red neuronal de regularización que se ha planteado como una herramienta para solucionar problemas generales de interpolación. Por esta razón, en el presente capítulo abordaremos, primero, la definición formal del problema de interpolación, luego plantearemos la estructura de una red de regularización para, finalmente, abordar con detenimiento la red de base radial, objeto de este apartado, incluyendo su arquitectura, modelo matemático, algoritmo de aprendizaje y aplicabilidad.

El problema de interpolación

Antes de abordar las redes RBF, es muy importante revisar el concepto de interpolación. La primera acepción que encontramos de interpolación, está relacionada con la estimación o búsqueda de nuevos valores a partir de un conjunto discreto de valores conocidos.

Otra acepción estrechamente ligada con el problema de la interpolación es la aproximación de una función de alta complejidad, normalmente con un gran número de variables independientes, por una más sencilla. Para el caso de redes neuronales, dicha función además de compleja es desconocida y sólo disponemos de un conjunto de datos experimentales o de simulación que la representan. A partir de estos datos, con un entrenamiento apropiado de la red, obtendremos una función que represente adecuadamente a estos datos conocidos. El siguiente paso es verificar la capacidad de generalización de la red con el fin de garantizar que la red aproxime los datos desconocidos que pertenezcan a la superficie que deseamos modelar.

Para ilustrar el problema de interpolación, partamos de dos conjunto puntos definidos, el primero en un espacio de *p* dimensiones y el segundo en un espacio de dimensión uno. El problema de interpolación consiste en encontrar una transformación matemática s tal que,

$$s : \Re^p \rightarrow \Re^1 \tag{6.1}$$

donde la transformación se puede visualizar como una hiper-superficie de dimensión *p+1*.

$$\Gamma \subset \Re^{p+1} \tag{6.2}$$

Por ejemplo, si suponemos que *p=1*, debemos encontrar una superficie de dimensión 2 para representar la interpolación, en la figura 6.1 tenemos un conjunto de datos conocidos representados por círculos y vemos como la línea a trazos lleva a cabo la interpolación.

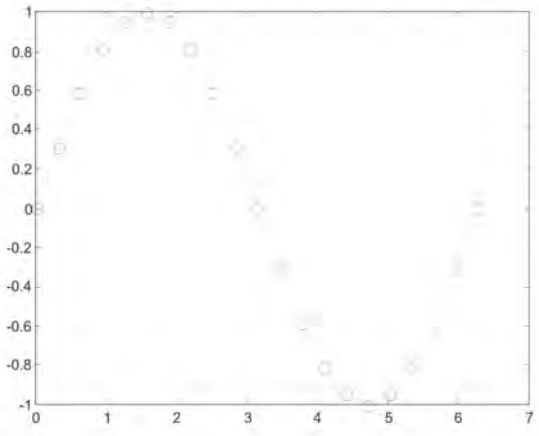


Fig. 6.1 Interpolación de un espacio de dimensión uno a otro de dimensión uno

En segundo ejemplo, supongamos que $p=2$, por lo que en este caso debemos encontrar una superficie de dimensión 3 para representar la interpolación, en la figura 6.2 partimos de un conjunto de datos conocidos representados por círculos y vemos como los asteriscos realizan la interpolación.

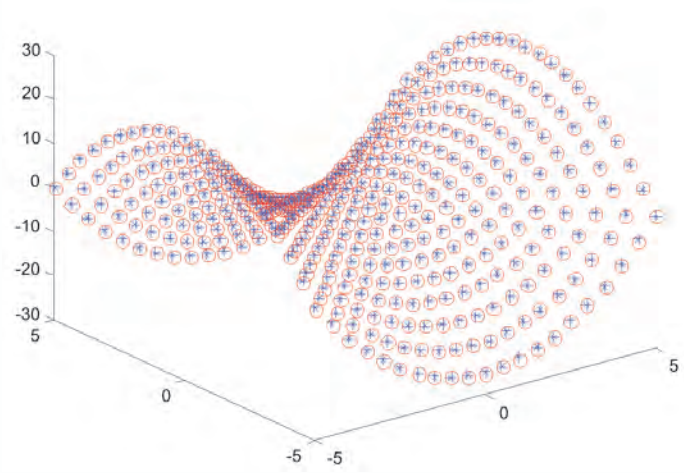


Fig. 6.2 Interpolación de un espacio de dimensión dos a otro de dimensión uno

El problema de interpolación hemos visto que se reduce a encontrar una función que aproxime al conjunto de puntos con un error mínimo. Si utilizamos redes neuronales artificiales para interpolar y el error cuadrático medio como medida de desempeño de la red, tal como lo vimos en el apartado 5.6 del Capítulo 3, la red puede entrar a memorizar los puntos de entrenamiento, que se manifiesta en una función con cambios abruptos y que tienen un mal desempeño ante puntos que estuvieron por fuera del entrenamiento.

Con el fin de evitar este problema de memorización adicionaremos a la función de error, un término de regularización que busca una función aprendida que no pase exactamente por todos los puntos pero que su comportamiento sea más suave, con el propósito de mejorar su capacidad de generalización. Sin embargo, el introducir un nuevo término de regularización trae como consecuencia que se presente múltiples soluciones al problema de interpolación.

En la ecuación 6.3, vemos como la función $\Phi(F)$ a minimizar en el proceso de aprendizaje tiene dos términos, $\Phi(E)$ que a su vez depende el Error Global de la red y $\Phi(F)$ introduce el concepto de regularización afectado por λ, con el fin de disponer un parámetro que nos permita modificar la suavidad de la función de interpolación.

$$\Phi(F) = \Phi(E) + \lambda\Phi(S) \tag{6.3}$$

En la ecuación 4, hemos sustituido la expresión del error global para mayor claridad de la expresión a minimizar.

$$\Phi(F) = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right)^2 + \lambda\Phi(S) \tag{6.4}$$

La solución al problema planteado en la ecuación 4, se puede expresar en términos de la funciones de Green *G,* definidas en términos de la norma euclideana, tal como observamos en la ecuación 6.5.

$$F(\mathrm{x}) = \sum_{i=1}^{N} w_i G(\|\mathrm{x} - \mathrm{x}_i\|) \tag{6.5}$$

Esta ecuación la podemos explicar gráficamente en forma de red neuronal, donde la función aprendida *F(x)* es igual a la sumatoria del producto de los pesos sinápticos de la red w_i por la salida de la función de Green cuando es evaluada con la norma que existe entre la entrada a la red y el centroide de la función de Green. La arquitectura presentada en la figura 6.3 se conoce como Red de Regularización, constituida por *N* entradas, una capa oculta con *L* neuronas, donde *L* es igual al número de datos disponibles en el conjunto de entrenamiento y una neurona de salida con función de activación lineal.

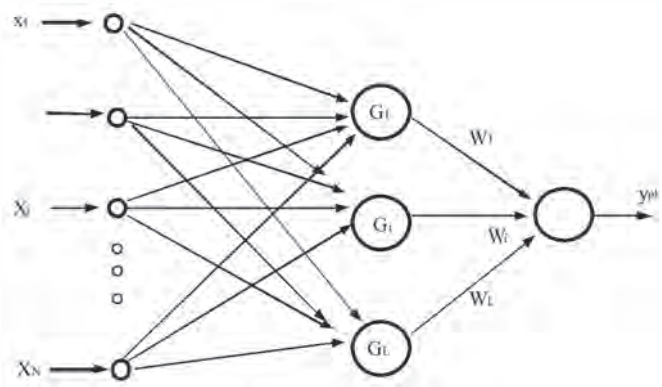


Fig. 6.3 Red de regularización

Las redes de regularización realizan una interpolación exacta, donde en la fase de entrenamiento el centro de cada una de las funciones de Green, se asocia a un dato del conjunto de entrenamiento, por lo que este parámetro ya queda definido. Para calcular los pesos w_i de la neurona de salida parti-

mos de la ecuación 6.6 que muestra la salida deseada de la red *d* en función de las salidas de las funciones de Green y de los pesos.

$$\mathbf{Gw = d} \tag{6.6}$$

Si la inversa de la matriz de funciones de Green existe, podemos obtener el vector de pesos ***w*** con la ecuación 7.

$$\mathbf{w = G^{-1}d} \tag{6.7}$$

Generalmente, las funciones de Green utilizadas son las de base radial como las mostradas en la figura 6.4.

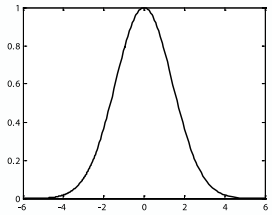
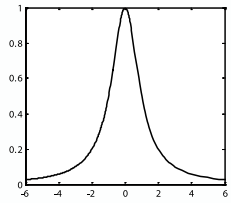
$\phi(x) = e^{-\frac{x^2}{2\sigma^2}}$  $\phi(x) = \left(x^2 + \sigma^2\right)^{-\alpha}$ 

Fig. 6.4 Funciones Base Radial

REDES DE BASE RADIAL

La solución del problema de interpolación exacta usando las redes de regularización, presenta los siguientes inconvenientes:

- El número de funciones de base es igual al número de puntos en el conjunto de datos, lo que claramente es inconveniente sobre todo en aplicaciones complejas cuyo conjunto de entrenamiento suele ser muy grande.
- La interpolación exacta de datos contaminados con ruido es típicamente una función oscilatoria.
- Debido a la proliferación de funciones de Green, cuando se tienen muchos datos de entrenamiento, se puede perder capacidad de generalización.

Estos inconvenientes los podemos salvar planteando las redes de base radial (RBF) gracias a las siguientes propiedades:

- El número de funciones base (*L*) no necesita ser igual al número de datos del conjunto de entrenamiento (*P*), por lo que en la arquitectura

de red RBF, L será mucho menor que P.

- Los centros de las funciones de base no coincidirán con los datos de entrenamiento y, por consiguiente, se convierten en parámetros a modificar en la etapa de aprendizaje de la red RBF.
- El ancho de la función de base también puede ser variable.

Arquitectura de una red de base radial

A primera vista, si revisamos la figura 6.5 no existe una diferencia clara entre la arquitectura de la red neuronal RBF y la del tipo MLP que vimos en sesiones pasadas. Como en el caso de la red neuronal tipo MLP con aprendizaje supervisado, la información fluye desde la capa de entrada hacia la salida de manera unidireccional. En la capa de entrada no existe procesamiento de la información y solo sirve de interfaz entre el mundo real y la RNA, transfiriendo el vector de entrada x a la capa oculta. La capa de salida toma la información proveniente de las neuronas de la capa oculta modificada por los pesos sinápticos existente entre estas dos capas y genera la salida total de la red, a esta neurona de salida se le incluye la entrada de umbral representada por w_0.

La diferencia esencial entre estas dos redes está en la capa oculta, empezando por la función de activación ϕ que ya no es como la que habíamos visto en los otros casos, sino que es una función del tipo base radial, normalmente la función de Gauss. Por esta razón, los parámetros que caracterizan a esta capa oculta son el centro de la función de activación (*centroide*) y su respectiva desviación estándar. Más adelante nos ocuparemos de profundizar en el procesamiento de esta capa oculta.

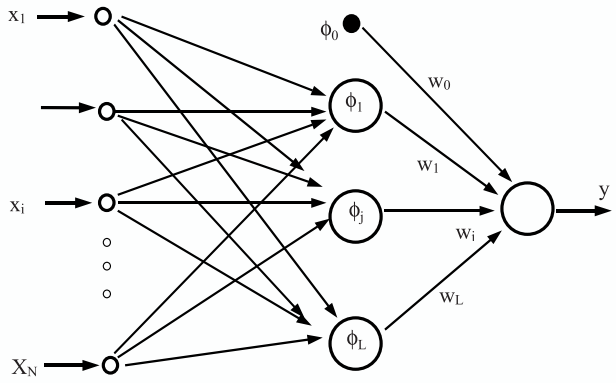


Fig. 6.5 Arquitectura de una red RBF

La capa de entrada corresponde simplemente a una interfaz entre el proceso a analizar o solucionar y la red neuronal. Toma los datos del entorno de trabajo de la red y los lleva a su interior para el procesamiento. La capa

de salida procesa los datos provenientes de la capa oculta y nos entrega la salida de la red.

En la capa oculta se establece la diferencia fundamental con la red tipo MLP, ya que no se calcula la entrada neta y aplica una función de activación sigmoide o lineal como en el caso del Perceptron multicapa, sino que opera con base en la distancia que existe entre el vector de entrada y el centroide de un elemento de la capa oculta, ecuación 8. En esta capa la función de activación es del tipo base radial, por ejemplo la función de Gauss. A esta distancia se le aplica una función de base radial, como por ejemplo la función de Gauss representada en la ecuación 9. La capa de salida procesa los datos provenientes de la capa oculta y como tiene una función de activación lineal la salida corresponde a la suma ponderada de las salidas que proporciona la capa oculta, ecuación 6.10.

$$r_j^2 = \left\| \mathbf{x} - \mathbf{c}_j \right\|^2 = \sum_i \left(x_i - c_{ji} \right)^2 \tag{6.8}$$

$$\phi(r) = e^{-r^2/2\sigma^2}$$
$$\phi(r) = \left(x^2 + \sigma^2 \right)^{-\alpha} \tag{6.9}$$

$$y(\mathbf{x}) = \sum_{j=1}^{L} \omega_j \phi_j(\mathbf{x}) + \omega_0 \tag{6.10}$$

En la figura 6.6, apreciamos el modelo de un elemento de procesamiento de la capa oculta que recibe el vector de entrada ***x*** de dimensión *N*, se calcula la norma respecto del vector centroide c_i, aplicamos la función de activación de base radial para generar la salida *y* de este elemento.

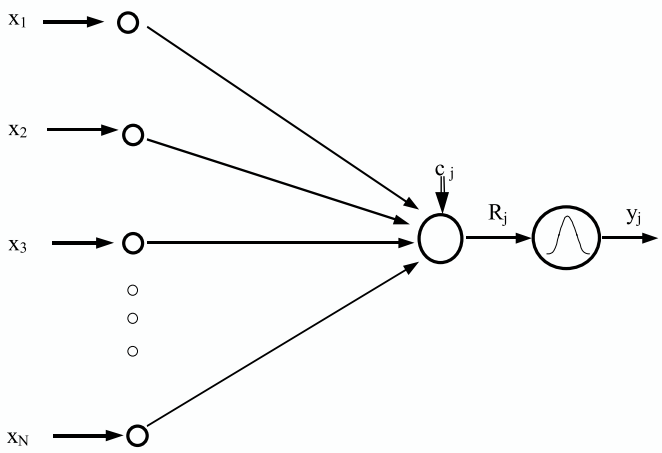


Fig. 6.6 Elemento de procesamiento de una RBF

En la figura 6.7, presentamos la arquitectura de una red *RBF* con *N* entradas, L elementos de procesamiento en la capa oculta con función de activación de base radial y un elemento de procesamiento en la salida con función de activación lineal.

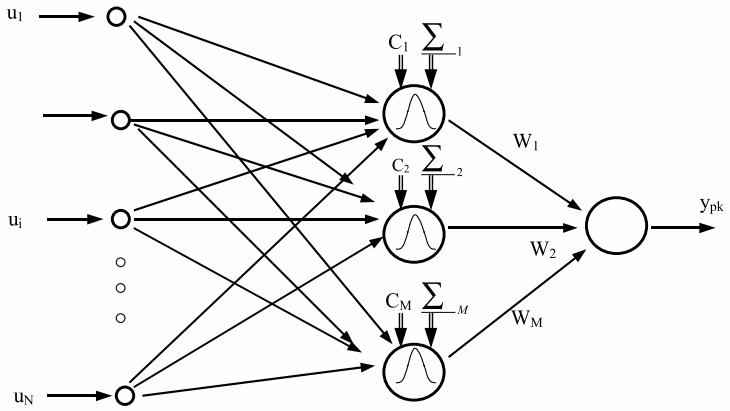


Fig. 6.7 Arquitectura de una red RBF

Entrenamiento de la red rbf

Con este tipo de red hemos planteado que el aprendizaje es híbrido por lo que incluye el aprendizaje supervisado en la capa de salida y el no-supervisado en la capa oculta; por esta razón el entrenamiento de las redes RBF lo dividiremos en dos etapas:

1. El entrenamiento de los elementos de procesamiento de la capa oculta lo realizaremos de forma No-Supervisada, de una manera muy similar a como entrenamos los mapas auto-organizados de *Kohonen*. Presentaremos los datos de entrada y los nodos de procesamiento de esta capa, las unidades de procesamiento ocultas se encargarán de dividirlos por sectores y cada una de las neuronas asumirán la descripción de los mismos en el espacio de entrada a través del vector *centroide*.
2. El entrenamiento de los elementos de procesamiento de la capa de salida lo realizaremos de forma Supervisada con el algoritmo de Corrección de Error, tal como fue descrito para el Perceptron.

Entrenamiento de la capa oculta

El aprendizaje No-Supervisado en estos elementos de procesamiento se cumple en las siguientes etapas:

1. Determinamos el valor de los centroides de los elementos ocultos y para ello utilizaremos el Algoritmo de las k-medias.
   Como esta fase es No-Supervisada no conocemos *a priori* el número de clases o sectores en que se dividen los datos, por lo que se convierte en un parámetro de diseño que depende del tipo de problema y su adecuada selección se hace de manera heurística.
   El paso siguiente es elegir los valores de los centroides C_j de los elementos de procesamiento de esta capa. La elección la podemos hacer de manera aleatoria, pero podemos tomar los primeros L patrones del conjunto de aprendizaje, donde L es el número de elementos de procesamiento ocultos escogido. Como podremos observar más adelante, el valor inicial de estos centroides es realmente irrelevante.
2. Luego de haber asignado el valor inicial de los centroides, en cada iteración t asignamos los patrones de aprendizaje x entre los L elementos de procesamiento ocultos. Cada patrón se asigna al elemento de cuyo centroide diste menos, con base en los siguientes pasos:
   - Tomamos el patrón x del conjunto de entrada y calculamos la distancia a cada uno de los centroides.
   - Verificamos cual es la menor de estas distancias.
   - El patrón de entrada se asigna al sector representado por el centroide cuya distancia haya sido mínima.
   - Repetimos este procedimiento para todos los patrones del conjunto de aprendizaje.
3. Calculamos los centroides de los nuevos sectores generados en el paso anterior, con base en la ecuación 6.11.

$$c_j = \frac{1}{p_j} \sum_{i=1}^{p_j} x_i \tag{6.11}$$

   p_j : Número de patrones que han correspondido a la neurona *j-ésima.*
4. Si hay variaciones en los centroides retornamos nuevamente al paso 2 en caso contrario, finalizamos la etapa de aprendizaje.
5. Calculamos con la ecuación 6.12 la desviación estándar σ_j por cada uno de los sectores encontrados, con el fin de determinar el radio de acción de cada una de los elementos de procesamiento.

$$\sigma_j^2 = \frac{1}{p_j} \sum_{i=1}^{p_j} \left\| x_i - c_j \right\|^2 \tag{6.12}$$

Entrenamiento de la capa de salida

El entrenamiento de los elementos de procesamiento de la capa de salida lo realizaremos de forma Supervisada con el algoritmo de Corrección de Error, tal como fue descrito para el Perceptron. Si conocemos la salida deseada d_{pk} y la salida que produce la red RBF y_{pk} podemos calcular el error global de la red con la ecuación 6.13, considerando a M como el número de elementos de procesamiento en la capa de salida y P la dimensión del conjunto de patrones de entrenamiento.

$$E_p = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right)^2 \tag{6.13}$$

Donde y_{pk} es la salida del *k-ésimo* elemento de procesamiento de la red, cuyo valor determinado por la ecuación 6.10, depende de las funciones de base radial; si re-escribimos esta incluyendo en la sumatoria el término de umbral, obtenemos la ecuación 6.14 y su representación matricial en la ecuación 6.15. Es bueno recordar, que el valor del umbral ϕ_0 es igual a 1.

$$y_{pk}(\mathbf{x}) = \sum_{j=0}^{L}\omega_{kj}\phi_j(r) \tag{6.14}$$

$$\mathbf{y}(\mathbf{x}) = \mathbf{w}\boldsymbol{\Phi} \tag{6.15}$$

De la ecuación 6.15 vemos la dependencia lineal entre los pesos de la capa de salida y la salida de la red, por lo que la matriz de pesos que minimiza el error cuadrático se puede encontrar usando el método de los mínimos cuadrados. Otra forma de calcular esta matriz de pesos es utilizando métodos basados en gradiente como los algoritmos ya estudiados en el capítulo 3, de Gradiente Descendente o Algoritmo de Levenberg Marquardt.

Diferencias entre las redes MLP y RBF

La clasificación de patrones es uno de los campos de mayor aplicación de las redes neuronales artificiales, en la figura 6.8 observamos la forma como llevan a cabo este proceso las redes tipo MLP y RBF.

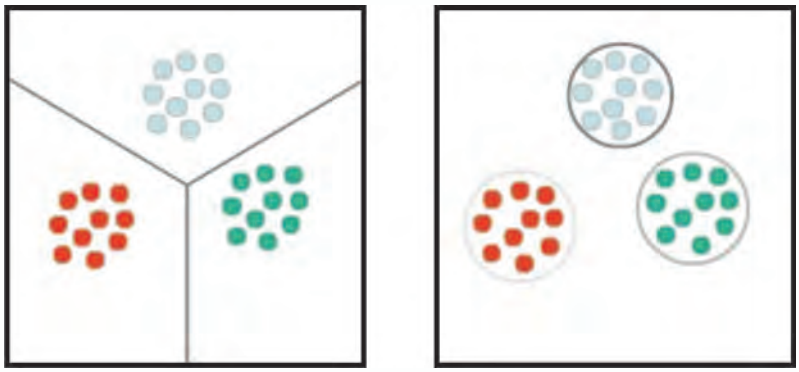


Fig. 6.8 Ejemplo de Clasificación de Patrones usando RNA

a) Clasificación usando MLP **b) Clasificación usando RBF**

Para solucionar este tipo de problemas las redes MLP de aprendizaje tipo *backpropagation*, tienen funciones de activación de respuesta en rango infinito y además las superficies que generan se pueden asimilar a hiperplanos en el espacio de decisión. Para el caso que presentamos en la figura 6.8.a, la información se representa en un espacio de dos dimensiones, la forma como éstas clasifican los datos de entrada es por medio de líneas rectas cuyas intersecciones definen semiplanos, en donde los datos pueden separarse por pertenencia o no a cada uno de ellos. De esta manera definimos tres clases de datos porque pertenecen a tres semiplanos que son delimitados por las líneas rectas.

En este ejemplo estamos limitados, por razones de dibujo, a un espacio de dos entradas; pero podríamos imaginarnos un espacio de *n* entradas, donde las superficies de separación son hiperplanos de *n-1* dimensiones, en donde igualmente se definen sub-espacios que nos separan adecuadamente las diferentes clases de datos existentes en el conjunto de entrenamiento.

Las redes tipo RBF tienen en su capa oculta funciones de activación de base radial, por lo que la separación de las clases es muy diferente al llevado a cabo por la red tipo MLP, tal como observamos en la figura 6.8.b, en ella vemos como se definen sectores alrededor de los datos de entrada, deberíamos pensar que el proceso es inverso, es decir, que los datos están distribuidos en el espacio de entrada de acuerdo a la información que albergan, y es la red la que busca esta organización y se adapta a ella a través del proceso de aprendizaje. Al final, la red RBF genera un número determinado de sectores que agrupan a los datos que poseen características similares en el espacio de entrada, llevando a cabo entonces el proceso de clasificación.

Para entender con más detalle lo acabado de exponer, veamos como una red tipo RBF resuelve el problema de la función lógica *AND* y comparemos con el resultado entregado por un Perceptron y una red tipo MLP cuando

resuelven este mismo problema.

El problema de la función lógica AND, un Perceptron lo resuelve generando una línea recta que separa los puntos con salida en cero (los azules) del punto con salida en uno (el rojo), tal como lo podemos apreciar en la figura 6.9.

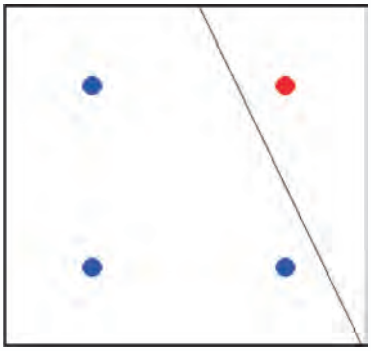


Fig. 6.9 Solución de la función AND con un Perceptron

Una red neuronal tipo MLP con funciones de activación sigmoidales en la capa oculta es capaz de generar una superficie de separación no-lineal para resolver este problema, esta superficie es de tal forma que los puntos con salida en cero queda en una región con salida cercana a cero y el punto rojo queda en una región con salida cercana a uno, tal como ilustramos en la figura 6.10.

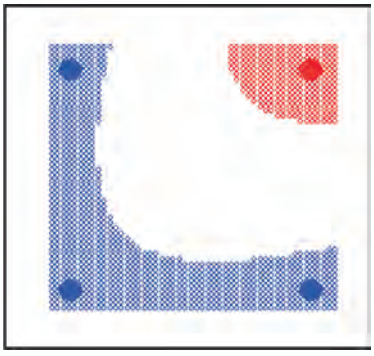


Fig. 6.10 Solución de la función AND con una red tipo MLP

Finalmente, una red tipo RBF para resolver el problema de función *AND*, como vemos en la figura 6.11 solo necesita una neurona gausiana en su capa oculta, dicha neurona se ubica en el punto con salida en uno, de tal manera que cuando a la red llega el patrón (1,1) la distancia de este patrón respecto al centro de la función de Gauss es mínima y, por ende, la salida es máxima, de tal manera que la salida de la red es máxima o en otras palabras tiende

a uno. Los otros patrones están más distantes del centro de la función de Gauss, por lo que su salida tiende a cero.

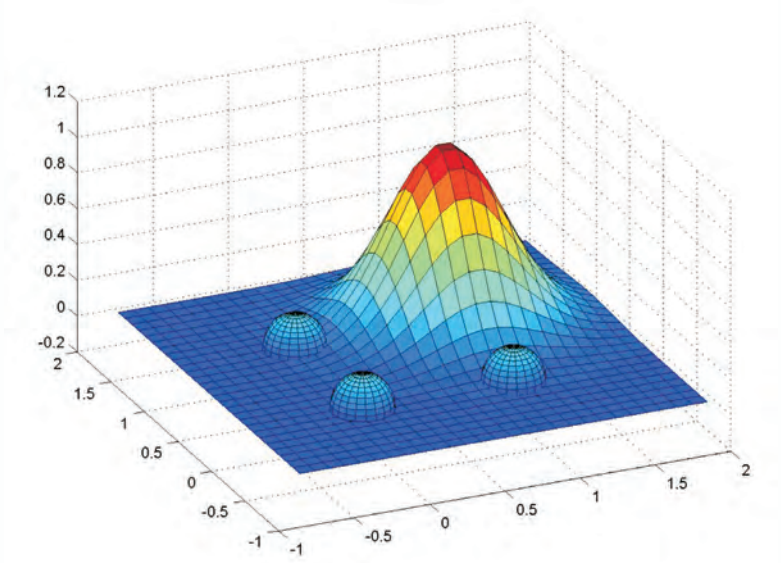


Fig. 6.11 Solución de la función AND con una red tipo RBF

Otro caso más interesante que ilustra las diferencias entre las redes neuronales tipo MLP y RBF, lo podemos ver cuando se resuelve el problema de la función lógica *XOR*. Una red tipo MLP con funciones de activación sigmoidales en la capa oculta es capaz de generar una superficie de separación no-lineal para resolver el problema de la XOR, tal como observamos en la figura 6.12, las salidas de los puntos de entrada (0,0) y (1,1) representados en el plano, se asocian a las superficies no lineales representadas en color azul que representa la salida 0; y las salidas de los puntos (1,0) y (0,1) se asocian a la superficie roja que representa la salida 1.

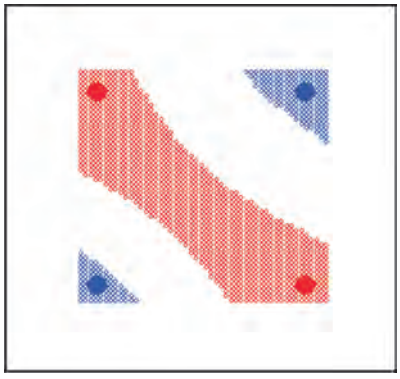


Fig. 6.12 Solución de la función XOR con una red tipo MLP

Una red tipo RBF para resolver este problema, necesita dos neuronas con función de activación gausianas en su capa oculta, dichas neuronas se ubican en los puntos con salida en uno, de tal manera que cuando a la red llega el patrón (0,1) o (1,0) la distancia de este patrón respecto al centroide de esta función es mínima y, por ende, la salida de la neurona es máxima. Los otros patrones están más alejados de los centroides y su salida tiende a cero, tal como observamos en la figura 6.13.

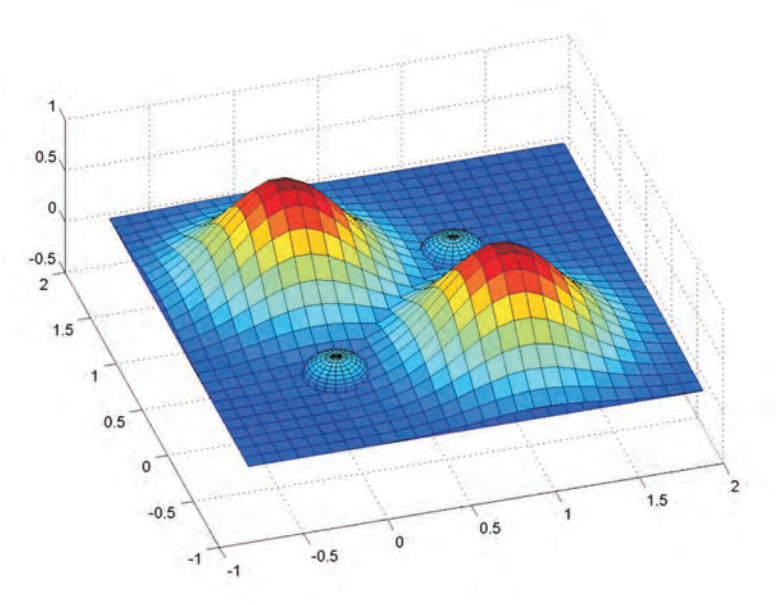


Fig. 6.13 Solución de la función XOR con una red tipo RBF

Para complementar el análisis previo, adicionalmente destacamos las siguientes diferencias que encontramos entre las redes neuronales tipo MLP y RBF:

- *Una red tipo MLP puede tener varias capas ocultas, mientras que en una RBF solo podemos definir una capa oculta.*
  Por definición de la arquitectura, las redes tipo RBF solo tienen una capa oculta debido a la naturaleza de procesamiento que tiene esta capa. En una red MLP es posible tener más de una capa oculta, de hecho en aplicaciones complejas una red MLP con dos capas ocultas puede resolver más rápidamente el problema que una MLP con una sola capa oculta.
- *El tipo de información de entrada a la capa oculta de una red RBF es diferente al que se procesa en una red MLP.*
  La información que llega a la capa oculta de una red tipo MLP es la

suma ponderada de las entradas de la red, mientras que en la red tipo RBF la capa oculta procesa la diferencia entre el vector de entrada y el vector centroide de las neuronas.

- *El procesamiento en la capa oculta en una red tipo MLP es global mientras en una red RBF es local.*
  Esta afirmación es una consecuencia del tipo de funciones de activación que estas redes tienen en su capa oculta, en una red tipo RBF la capa oculta tiene funciones de Gauss cuya activación se encuentra localizada alrededor de su centro donde la salida es máxima, mientras que las redes tipo MLP tienen funciones de activación lineales o sigmoidales en donde la salida máxima se da para un rango mucho mayor de entradas.

Aproximación práctica

Ejemplo de interpolación exacta con MATLAB®

Este capítulo lo iniciamos con una revisión a los conceptos básicos de interpolación exacta, y ahora la primera propuesta práctica es poder llevar a cabo un ejercicio para ilustrar este tipo de aplicación a partir de la función descrita en la ecuación 6.16, cuyos datos de salida están contaminados con ruido blanco centrado en cero y desviación estándar 0.5.

$$y = 0.5 + 0.4\sin(2\pi x) \tag{6.16}$$

La interpolación exacta la vamos a llevar a cabo con una red de regularización con funciones de Gauss para la activación de las neuronas de la capa oculta. Los centros de estas funciones los tomamos, inicialmente, iguales a los patrones de entrada y su desviación estándar es igual a 0.067. El código en MATLAB® que implementa la aplicación lo presentamos a continuación, y debe prestarse especial atención a la forma como se lleva a cabo la inversión de matrices.

```
% Ejemplo de interpolación exacta
close all;
clear;
figure
x=0:1/29:1;
y=0.5+0.4*sin(2*pi*x);
ruido=normrnd(0,0.05,1,length(y));
yruido=y+ruido;
plot(x,y,'--b',x,yruido,'ob');
hold on;
```

```
sigma=0.067;

cx=x;
for i=1:30
   dis=x(i)*ones(1,length(cx))- cx ;
   Phi(i,:)=exp( - ( dis./(2*sigma)).^2 );
end;

% No funciona  a pesar de ser la manera más intuitiva de hacer
la
% Operacion matricial
% W=inv(Phi)*yruido';

% La manera correcta de hacer el cálculo es con la división izqui-
erda, pues de esta
 %manera MATLAB realiza una estimación por mínimos cuadra-
dos de W

W=(Phi)\yruido';
xeval=0:1/290:1;
N=length(xeval);
for i=1:N
dis= xeval(i)*ones(1,length(cx))- cx ;
yco=exp( - ( dis./(2*sigma)).^2 );
yred(i)=(yco)*(W);
end;

plot(xeval,yred,'r'), axis([0 1 0 1.25])
hold off;
```

Finalmente, en la figura 6.14 apreciamos el resultado de hacer una interpolación exacta, donde claramente observamos que la capacidad de generalización de esta red es pobre, toda vez que la línea a trazos representa la función sin contaminación por ruido.

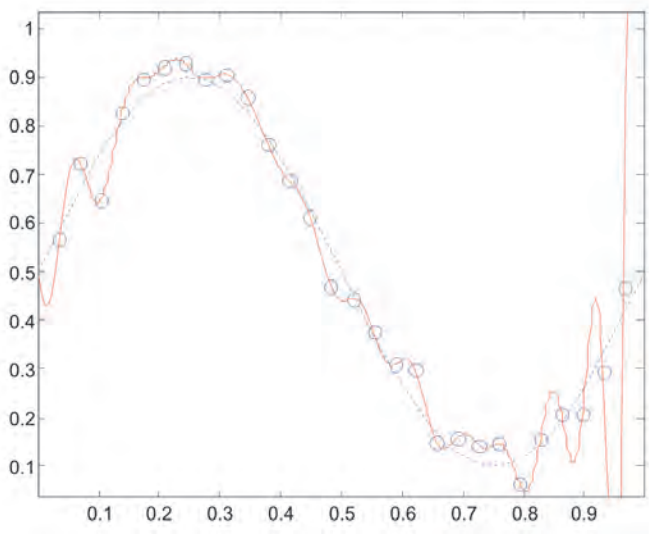


Fig. 6.14 Interpolación exacta realizada con una función de Gauss

Aprendizaje de la función xor

El código MATLAB® que presentamos a continuación nos resuelve el problema de la función lógica *XOR* usando una red RBF. El propósito es ilustrar como este tipo de red puede solucionar problemas no linealmente separables y motivar su estudio para su aplicación en problemas de mayor complejidad.

```
%Solución al problema de la XOR usando una red tipo RBF
close all;
x=[0 0 1 1;
  0 1 0 1];
yd=[0 1 1 0];
red=newrb(x,yd,0,1);
Yred=Sim(red,x);
clc;
disp('La salida de la red entrenada es')
disp(Yred);
```

Aprendizaje de una función de una variable

Con la red neuronal tipo RBF, a diferencia de la red de regularización, ya no hacemos un proceso de interpolación exacta cuando se entrena para aprender una función y con esto ganamos en capacidad de generalización. En este ejercicio de aplicación, buscamos aprender la función descrita con

la ecuación, el objetivo buscado es aprender la función que presentamos en la ecuación 6.17 y cuya descripción gráfica se aprecia en la figura 6.15.

$$f(x) = \text{sen}(x)\cos(2*x) \tag{6.17}$$

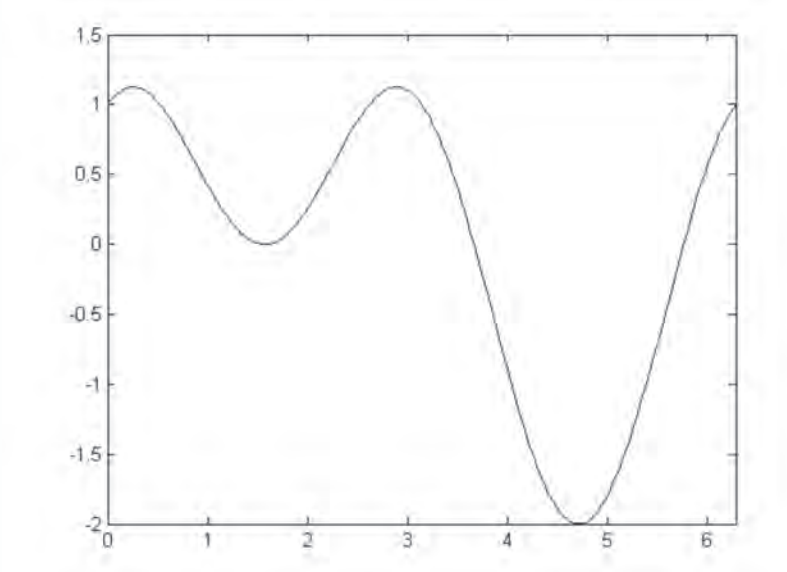


Fig. 6.15 Función a Aproximar con la red tipo RBF

A continuación presentamos un programa escrito para MATLAB® que permite alcanzar el objetivo planteado.

```
%Ejemplo de interpolación usando una red tipo RBF

close all;
x=0:0.2:2*pi;
y=sin(x)+cos(2*x);

% Parámetro de desviación estándar o ancho de la función de
activación de las neuronas de la %capa oculta.
des=1;
red=newrb(x,y,0.002,des);
xval=0:0.1:2*pi;
yred=sim(red,xval);
yval=sin(xval)+cos(2*xval);
figure;
plot(x,y,'ok',xval,yred,'k',xval,yval,'--k');
title('Azul = Original Rojo = Red')
```

Al entrenar la red con este programa, obtenemos una respuesta similar a la de la figura 6.16, donde la línea azul representa la función a aproximar y los círculos rojos corresponden a la salida de la red. Claramente se observa que el proceso de aprendizaje es satisfactorio.

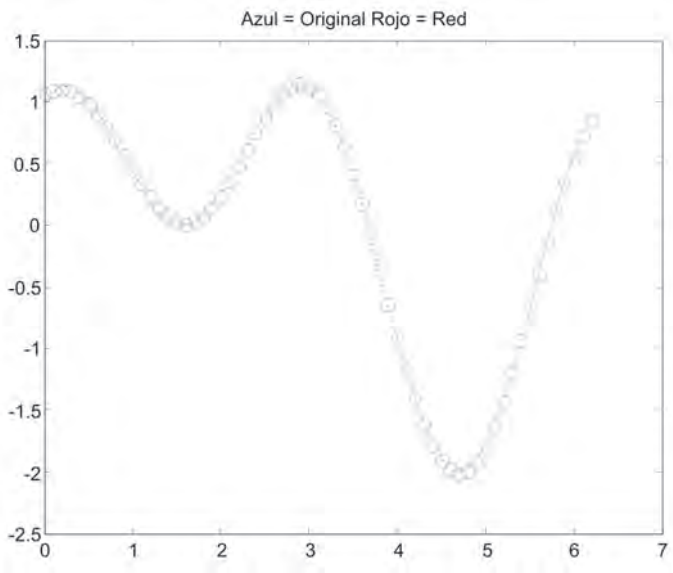


Fig. 6.16 Patrones de entenamiento y Salida de validación de la Red (des=1)

Un aspecto fundamental para la aproximación de funciones con redes RBF, es el parámetro de desviación estándar de la función de Gauss que definimos para las neuronas de la capa oculta. Si el valor de la desviación estándar es muy alto, por ejemplo 100, la salida de la red tiende a generar una interpolación excesivamente suave, tal como apreciamos en la figura 6.17.

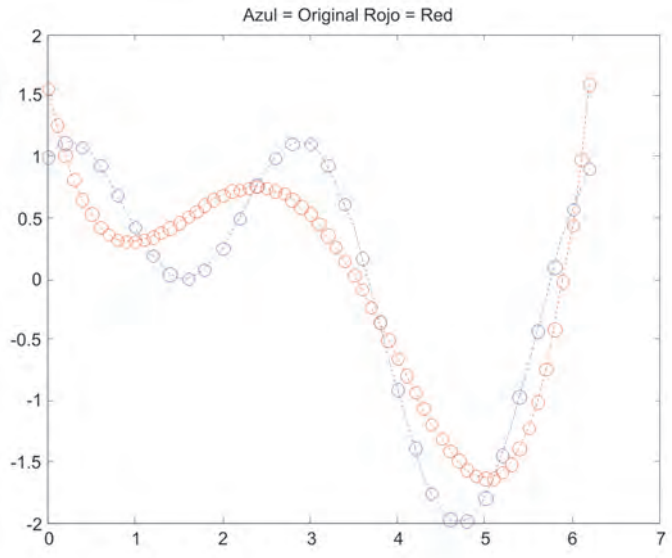


Fig. 6.17 Interpolación cuando la desviación estándar es alta

Si por el contrario el parámetro de desviación se hace muy pequeño, por ejemplo 0.1, la red presenta problemas de sobre entrenamiento y pierde capacidad de generalización, tal como se aprecia en la figura 6.18.

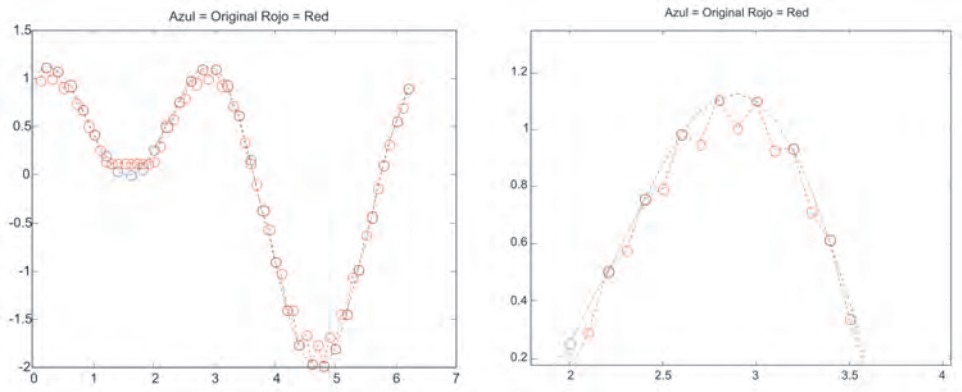


Fig. 6.18 a) Interpolación cuando la desviación estándar es baja
b) Acercamiento para visualizar el sobre-entrenamiento

Identificación de la dinámica de un sistema con una red rbf

En este proyecto asumiremos un reto muy interesante que se soporta en la propiedad de estas redes, de aproximar una función con una precisión predefinida, el objetivo es realizar la identificación de un sistema a partir

de datos experimentales de entrada y salida, que utilizaremos como datos de entrenamiento de la red. El proceso para identificar un sistema dinámico usando redes RBF es similar al utilizado con las redes tipo MLP, que describimos ampliamente en la sección 6.5 del capítulo 3.

En primera instancia vamos a identificar una planta lineal, cuya función de transferencia conocemos y está descrita con la ecuación 6.18.

$$G(s) = \frac{0.5}{(s + 0.5)} \tag{6.18}$$

Diseño del experimento y muestreo de datos

Primero es necesario definir el rango en el que se desea identificar la planta, en este ejemplo el rango de los datos de entrada es [0,1]. Usando la herramienta *simulink* de *MATLAB*® podemos simular el sistema con el esquema de la figura 6.19, obtenemos el conjunto de datos para el entrenamiento de la red.

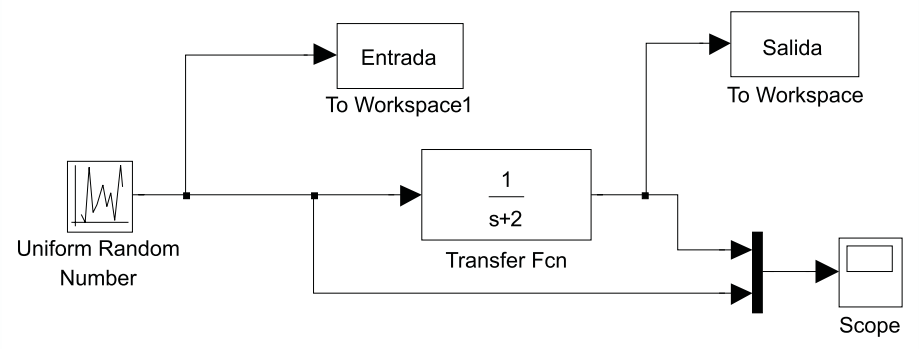


Fig. 6.19 Esquema en Simulink del experimento

El bloque de entrada es un bloque tipo *Band-Limited White Noise* que se encuentra en la biblioteca de *sources*.

El bloque *offset* es un bloque tipo *constante* también de la biblioteca *sources.*

Debemos ajustar los parámetros de estos bloques de manera adecuada, de tal manera que obtengamos datos de entrada y salida como los de la figura 6.20.

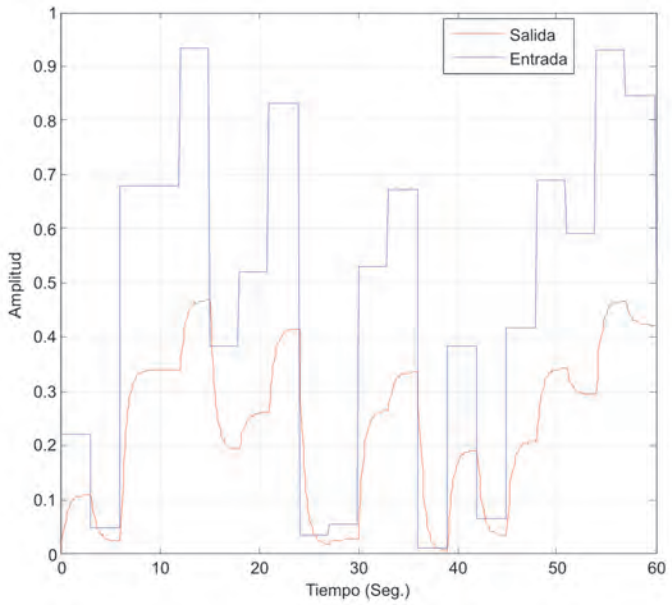


Fig. 6.20 Una posible entrada y su salida obtenida en la Simulación

Recomendaciones:

- El tiempo de muestreo lo debemos ajustar entre 0.1 y 0.2 veces la constante de tiempo más rápida del sistema.
- El ancho de los escalones de entrada debe ser aproximadamente el tiempo de estabilización de la planta.
- La entrada del experimento debe cubrir todo el rango posible de valores de la variable de entrada del proceso.
- Por lo general la entrada del experimento se diseña para que cubra el rango de la entrada del proceso ± el 20%.

Modelo a usar y estimación de parámetros (entrenamiento de la red)

Con los datos experimentales, procedemos a definir una arquitectura de red que sirva para identificar la planta en cuestión. Como la planta es de primer orden y al usar un modelo ARX, se necesitan como entrada de la red, un retardo de la entrada y un retardo de la salida para obtener la salida actual; esto es justamente lo que aplicamos en el regresor del ecuación 6.19.

$$y(k) = f(u(k-1), y(k-1))) \tag{6.19}$$

Lo anterior nos lleva a la siguiente arquitectura de red.

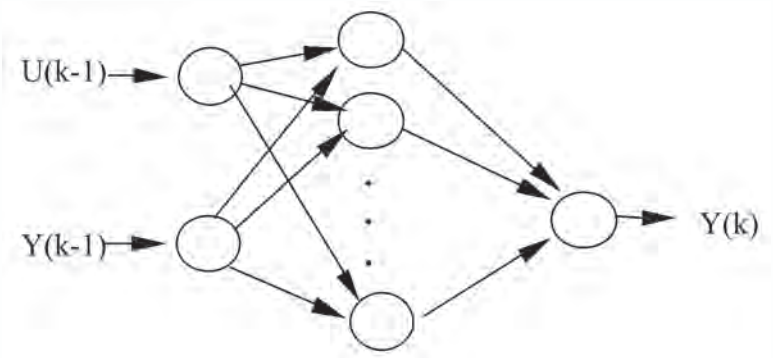


Fig. 6.21 Estructura de la red utilizada para la identificación

Como vemos en la figura 6.21, la red tiene dos elementos en la capa de entrada, en la capa oculta el número de elementos de procesamiento lo define el usuario en la aplicación software que entregamos en el recuadro, y elemento de procesamiento en la capa de salida.

El paso siguiente es organizar los datos que se obtuvieron durante el experimento y definir los patrones de entrenamiento a utilizar en el proceso de aprendizaje de la Red Neuronal Artificial. Esto lo llevamos a cabo construyendo un **.m* en MATLAB®, el cual lo denominaremos *entrenar.m* y aparte de organizar los vectores para obtener las entrada que necesita la red, también realiza el entrenamiento de la red.

Creemos el archivo *entrenar.m* usando el código MATLAB® que se muestra a continuación.

```
% Entrenamiento de una red MLP para la identificación
% de una planta lineal de primer orden
% u debe contener los datos de entrada a la planta.
% y debe contener los datos de salida a la planta.
% u,y deben son vectores columna.

%|  Entrada1  | Entrada2   | Salida  |
%=====================================
====
%| U(1 )       |  Y(1)      | Y(2)  |
%| U(2 )       |  Y(2)      | Y(3)  |
%| ..................................................................................
%|   U(K-1)   |  Y(K-1)  | Y(K)  |
%
% Se pierde el último dato del vector de entrada U
```

```
close all;

U=Entrada';
Y=Salida';

N=length(U);
X=[U(1:N-1);
   Y(1:N-1)];
Yd=[Y(2:N)];

red=newrb(X,Yd,0.0001,4)
```

Validación del modelo obtenido con la red

Luego de procesar el entrenamiento, se tiene en el objeto *red,* una red que ha aprendido la dinámica de la planta, para validar este modelo se realiza un esquema en la herramienta *Simulink* de MATLAB® como el mostrado en la figura 6.22.

Primero debe obtenerse una representación en diagramas de bloques de la red neuronal que se ha acabado de entrenar, para esto se utiliza el comando *gensim* indicando el tiempo de muestreo usado para la obtención de los datos.

```
>> gensim(red,Ts)
```

donde,

red : Objeto de la red que se ha entrenado
Ts : Tiempo de muestreo seleccionado

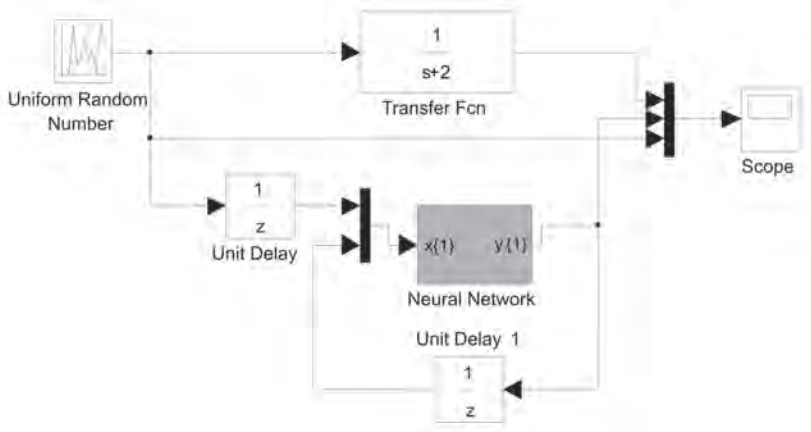


Fig. 6.22 Esquema en Simulink para validar el modelo

Cuando llevamos a cabo la simulación del sistema en *Simulink*, el programa nos entrega una salida como la mostrada en la figura 6.23. Como podemos observar la salida de la red sigue plenamente la salida de la planta original, por lo que aseguramos que la red neuronal tipo MLP aprendió o identificó la dinámica de la planta.

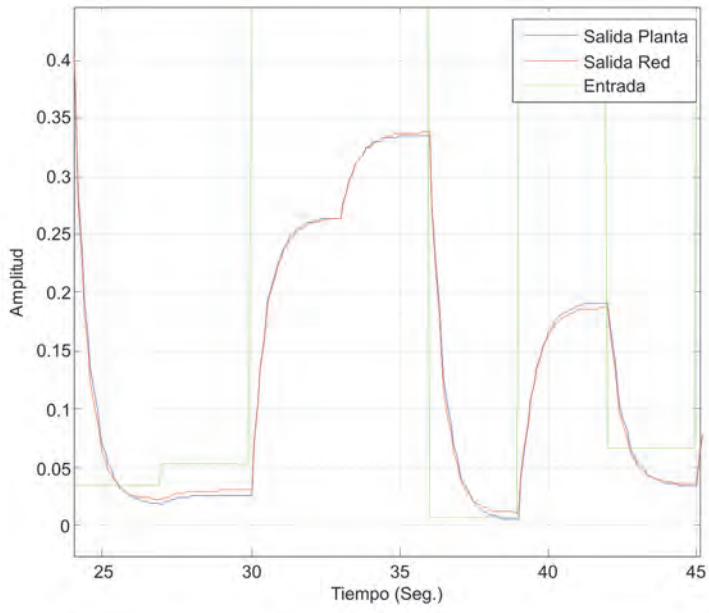


Fig. 6.23 Validación del modelo obtenido

Proyectos propuestos

1. Identifique la siguiente planta usando una red neuronal tipo RBF implemen-tada en MATLAB®. Realice una comparación de resultados con respecto a la solución obtenida con un Perceptron Multicapa en el proyecto 6 del capítulo 3 de este libro.

$$G(s) = \frac{5}{(s^2 + 4s + 10)}$$

2. Entrene una red tipo RBF tanto en MATLAB®, que aprenda la función *(0.5*sin(x)+0.5*sin(2*x))* de tal forma que x esté en el rango $-\pi < x < \pi$. Para validar el entrenamiento verifique la capacidad de generalización de la red entrenada.
3. Entrene una red tipo RBF en MATLAB® que aprenda la función

Seno(x)/x de tal forma que *x* esté en el rango $-10 < x < 10$. Para validar el entrenamiento verifique la capacidad de generalización de la red entrenada.

4. Entrene una red neuronal tipo RBF en MATLAB® que sirva para reconocer las vocales.
5. Entrene una red neuronal tipo RBF en MATLAB® que sirva para reconocer los dígitos del 0-9.
6. Entrene una red tipo RBF en MATLAB® que aprenda la siguiente función de dos variables que se muestra en la figura 6.24. Verifique la capacidad de generalización de la red entrenada.

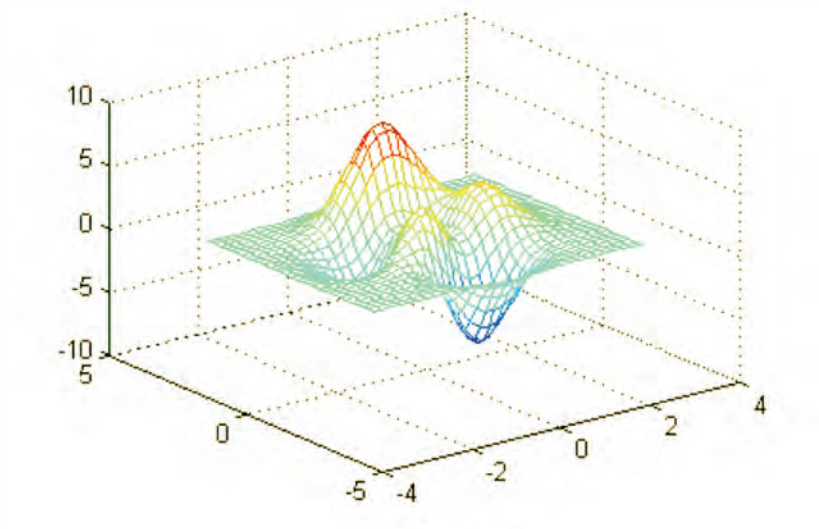


Fig. 6.24 Función de dos variables a identificar

Sugerencia: Utilice el siguiente código para generar la gráfica de la función a aprender.

```
Xini=[-3:0.2:3]; Yini=Xini;
[x,y]=meshgrid(Xini,Yini);
z= 3*(1-x).^2.*exp(-(x.^2) – (y+1).^2) ...
10*(x/5 – x.^3 – y.^5).*exp(-x.^2-y.^2) ...
1/3*exp(-(x+1).^2 – y.^2);
mesh(x,y,z)
```