Universidad de Alcalá

PRÁCTICA 1: INDENTIFICACIÓN Y CONTROL NEURONAL

SISTEMAS DE CONTROL INTELIGENTE GII
CURSO 2023/24

JAVIER JOSÉ GUZMÁN RUBIO 09246617L
LUCAS ÁLVAREZ RODRÍGUEZ 09109677K

Contenido

PARTE 1

Ejercicio 1. Perceptrón

Se desea clasificar un conjunto de datos pertenecientes a cuatro clases diferentes. Los datos y las clases a las que pertenecen con los que se muestra a continuación:

x_1	x_0	Clase
0.1	1.2	2
0.7	1.8	2
0.8	1.6	2
0.8	0.6	0
1.0	0.8	0
0.3	0.5	3
0.0	0.2	3
-0.3	0.8	3
-0.5	-1.5	1
-1.5	-1.3	1

Se desea diseñar un clasificador neuronal mediante un perceptrón simple que clasifique estos datos. Diseñe el clasificador, visualice los parámetros de la red y dibuje los datos junto con las superficies que los separan.

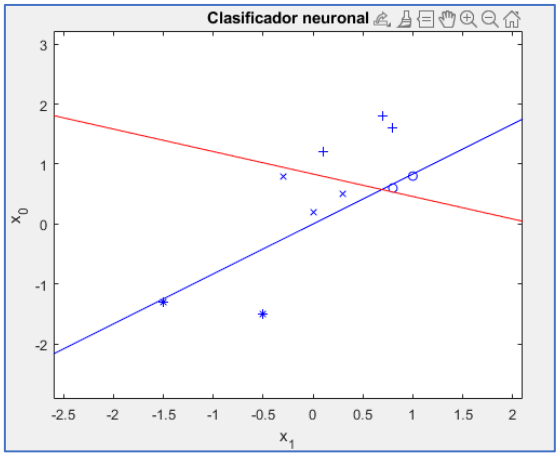
Se parte de la siguiente estructura:

```
%% Se declaran las matrices P y T con los valores:

P = [0.1 0.7 0.8 0.8 1.0 0.3 0.0 -0.3 -0.5 -1.5;
     1.2 1.8 1.6 0.6 0.8 0.5 0.2 0.8 -1.5 -1.3];
T = [1 1 1 0 0 1 1 1 0 0;
     0 0 0 0 0 1 1 1 1 1];
```

Se crea una red perceptrón con los valores anteriores:

```
% Se crea una red perceptron con estos valores y se entrena

net = newp(P, T);
net = train(net, P, T);

Y = sim(net, P);
```

Resultado:

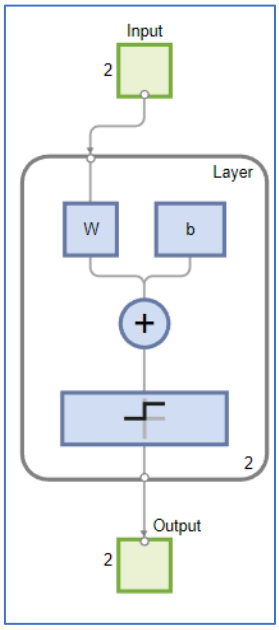


Cada uno de los puntos que se pueden observar en la gráfica son los parámetros de entrada. El eje de abscisas mide su valor en la columna x_1, mientras que el eje de ordenadas el de x_0.

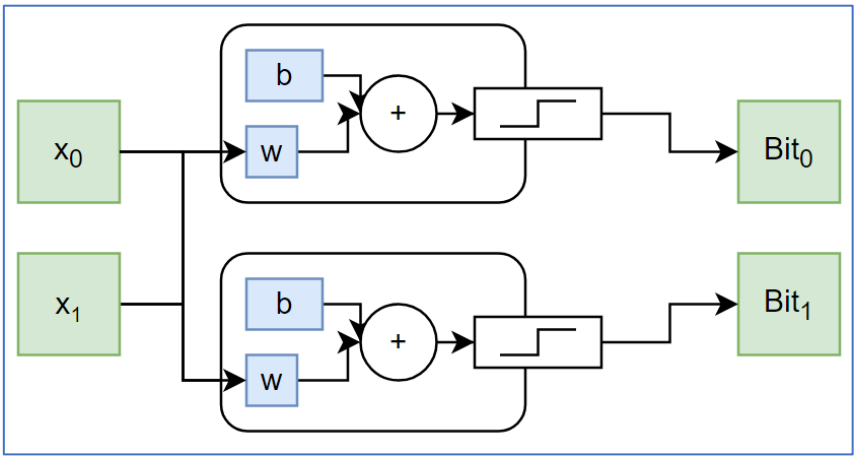
Los puntos que se encuentran ubicados por encima de la recta azul y la roja son los pertenecientes a la clase 2, representados con el símbolo +; los que están por encima de la azul y debajo de la roja, son los de la clase 3, representados por el símbolo x; los puntos por encima de la recta roja y debajo de la azul, son los pertenecientes a la clase 0, de símbolo ○, y los puntos por debajo de la recta roja y azul son los de la clase 1, que están representados con *.

¿Consigue la red separar los datos?, ¿cuántas neuronas tiene la capa de salida?, ¿por qué?

Sí, como se ha comentado en el resultado anterior, se separan en función de la clase a la que pertenecen. Es posible obtener un diagrama de la red, presentado a continuación:



Como se puede observar, el perceptrón cuenta con una entrada de tamaño 2, que, como se ha comentado anteriormente, son las columnas x1 y x0 en forma de vector bidimensional. La salida es de tamaño 2, puesto que son necesarios 2 dígitos binarios para representar las 4 clases posibles. Por ende, al tratarse de una red perceptrón (cuyas neuronas tienen una función de salida escalonada), se necesitan 2 de estas para aportar estos valores binarios de clasificación. Se muestra un esquema más ilustrativo a continuación:



¿Qué ocurre si se incorpora al conjunto un nuevo dato: [0.0 -1.5] de la clase 3?

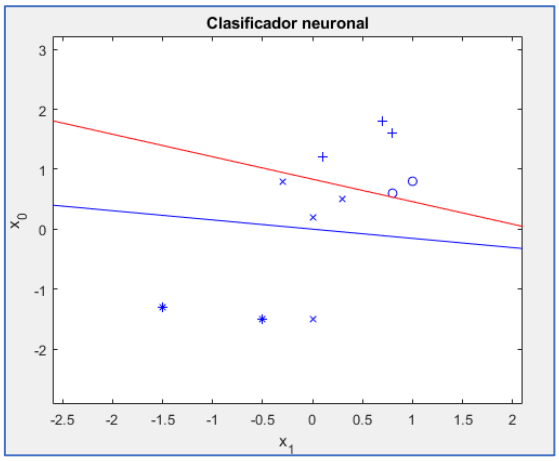
Si se introduce un nuevo dato, la estructura será la siguiente:

```
%% Se incorpora el nuevo dato
P2 = [0.1 0.7 0.8 0.8 1.0 0.3 0.0 -0.3 -0.5 -1.5 0.0;
     1.2 1.8 1.6 0.6 0.8 0.5 0.2 0.8 -1.5 -1.3 -1.5];
T2 = [1 1 1 0 0 1 1 1 0 0 1;
     0 0 0 0 0 1 1 1 1 1 1];
```

Para este nuevo dato introducido y con la nueva estructura, se creará un nuevo perceptrón.

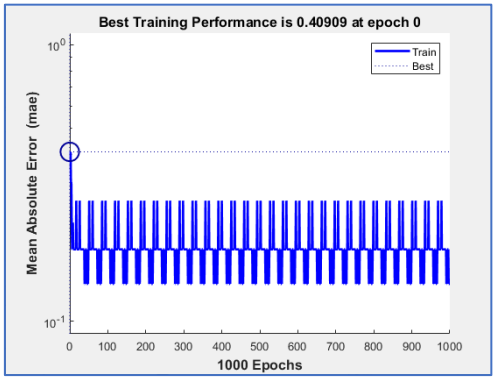
```
net2 = newp(P2, T2);
net2 = train(net2, P2, T2);

Y2 = sim(net2, P2);
```

Como resultado de este, se obtendrá:



En esta nueva gráfica, ya no se observa la división por clases presente para el primer conjunto de datos. Los datos de la clase 2, están con los de la clase 0, mientras que los de la clase 3 se encuentran por debajo la recta roja y encima de la azul, al igual que sucedía con el primer conjunto de datos, exceptuando uno de ellos que se encuentra junto con los de la clase 1 los cuales permanecen debajo de la recta roja y azul. La única excepción a esta distribución es la del nuevo valor añadido, [0, -1.5], de clase 3.

Este valor debería, según el criterio establecido por el Perceptrón anterior, ser clasificado como de clase 1. Sin embargo, al haberse empleado en el conjunto de datos de entrenamiento, se considera facto. Al no existir un par de funciones lineales capaz de clasificar este nuevo conjunto, la red trata de aproximarse por reducción a la mejor clasificación posible. Esto da como resultado un clasificador incorrecto, incluso contradictorio. Observar el gráfico de performance corrobora esta teoría:



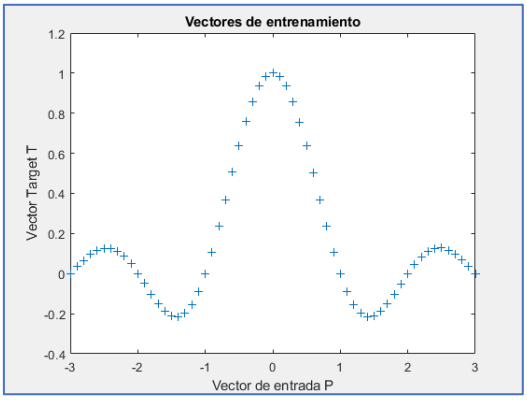
Ejercicio 2. Aproximación de funciones

Una de las aplicaciones inmediatas de las redes neuronales es la aproximación de funciones. Para ello, Matlab dispone de una red optimizada, fitnet, con la que se trabajará en este ejercicio. El objetivo en este caso es aproximar la función $f = sinc(t)$ tal y como se muestra a continuación:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% APROXIMACIÓN DE FUNCIONES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all; close all;

%    DEFINICIÓN DE LOS VECTORES DE ENTRADA-SALIDA
%    ============================================

t = -3:.1:3; % eje de tiempo
F=sinc(t)+.001*randn(size(t)); % función que se desea aproximar

plot(t,F,'+');
title('Vectores de entrenamiento');
xlabel('Vector de entrada P');
ylabel('Vector Target T');

%    DISEÑO DE LA RED
%    ================

hiddenLayerSize = 4;
net = fitnet(hiddenLayerSize,'trainrp');

net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

net = train(net,t,F);

Y=net(t);

plot(t,F,'+'); hold on;
plot(t,Y,'-r'); hold off;
title('Vectores de entrenamiento');
xlabel('Vector de entrada P');
ylabel('Vector Target T');
```

Estudie los efectos sobre la solución final de modificar el método de entrenamiento (consulte la ayuda de Matlab y pruebe 4 métodos diferentes) y el número de neuronas de la capa oculta.

La función que se desea aproximar, representada gráficamente, es la siguiente:



Para las aproximaciones, se van a utilizar los siguientes métodos de entrenamiento:

- Resilient Backpropagation: trainRP
- Levenber-Marquardt: trainLM
- BFGS Quasi-Newton: trainBFG
- Gradient Descent: trainGD

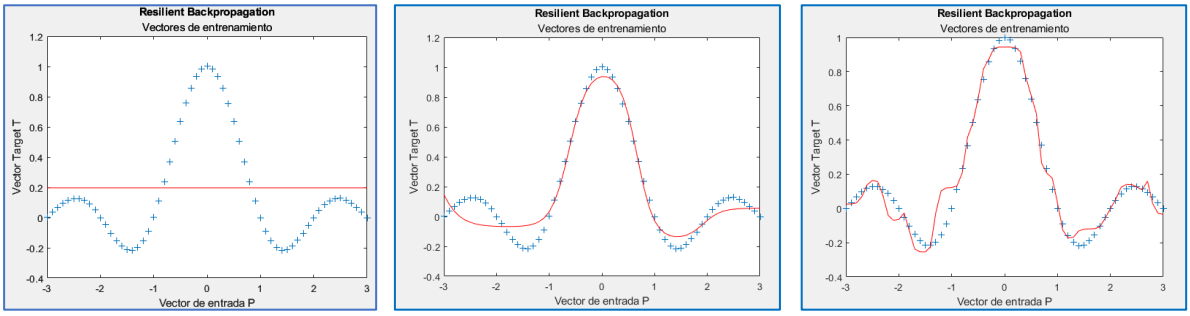
Sobre cada uno de estos métodos de entrenamiento, se especificarán tres cantidades de neuronas en la capa oculta de la red para comparar el resultado: 1, 5 y 25 neuronas.

1. Resilient Backpropagation

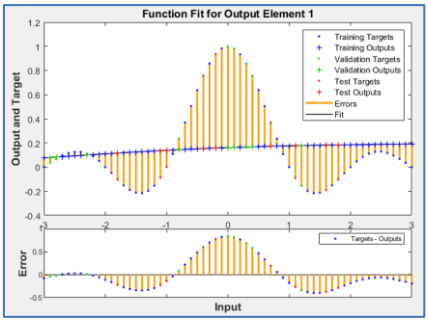
El parámetro ‘trainrp’ permite a la función *fitnet* formar una nueva red neuronal que siga este algoritmo de entrenamiento:

```
%% TRAINRP
% ==================
hiddenLayerSize = 5; % número de neuronas en la capa oculta

netRP = fitnet(hiddenLayerSize,'trainrp');

%Dividir los datos
netRP.divideParam.trainRatio = 70/100;
netRP.divideParam.valRatio = 15/100;
netRP.divideParam.testRatio = 15/100;
netRP = train(netRP,t,F);
YRP=netRP(t);

figure('Name','TRAINRP','NumberTitle','off') %Titulo de la ventana
plot(t,F,'+'); hold on;
plot(t,YRP,'-r'); hold off;
title('Resilient Backpropagation');
subtitle('Vectores de entrenamiento');
xlabel('Vector de entrada P');
ylabel('Vector Target T');
```

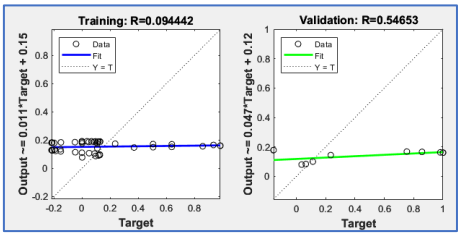
Utilizando redes de este tipo, de 1, 5 y 25 neuronas en la capa oculta, producen los siguientes resultados, respectivamente:



- **1 neurona:** Como es de esperar, una única neurona en la capa oculta resulta en un ajuste prácticamente nulo. Realmente no se puede comentar mucho sobre las medidas de la red, ya que ha dado una línea horizontal.

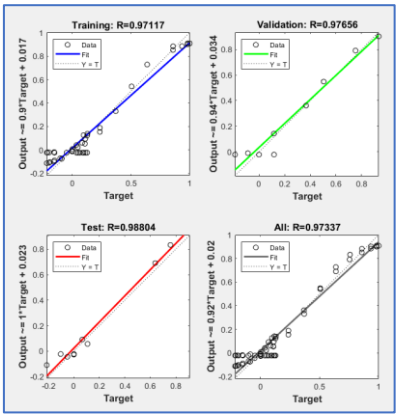


Quizá resulta ilustrativo señalar que, al ser un valor común en todas sus predicciones, los gráficos de error forman una figura similar a la que dibujan los datos.



Las rectas de regresión reflejan la nula adaptación a esta función.

- **5 neuronas:** Incrementar el número de neuronas de la capa oculta permite a la red ajustarse con mayor precisión a la función a predecir. Se puede apreciar que, con esta cantidad, la red encuentra dificultades con los máximos y mínimos locales de la función.



Las rectas de regresión muestran una situación bastante más positiva con respecto a las predicciones de la red.

- **25 neuronas:** Con esta cantidad, la red comienza a "anticiparse" a los valores de la función, como se señalaba antes, cuando se acercan a puntos de inflexión.

Como es de esperar, aumentar, hasta cierto punto, el número de neuronas resulta beneficioso para la capacidad de predicción de la red. Los valores óptimos de la red con esta última configuración son:

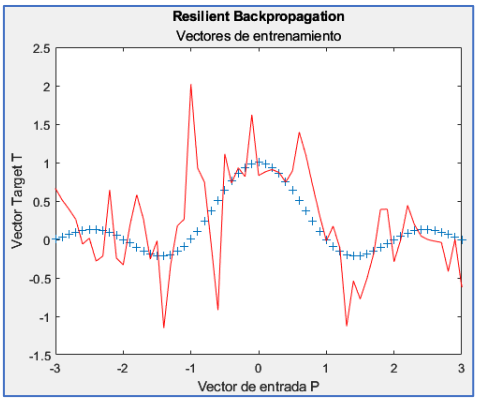
Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	22	1000
Elapsed Time	-	00:00:00	-
Performance	1.84	0.00939	0
Gradient	4.3	0.0699	1e-05
Validation Checks	0	6	6

Training Algorithms

Data Division: Random dividerand
Training: RProp trainrp
Performance: Mean Squared Error mse
Calculations: MEX

Sobrepasar este número de neuronas óptimo puede resultar en modelos erráticos:

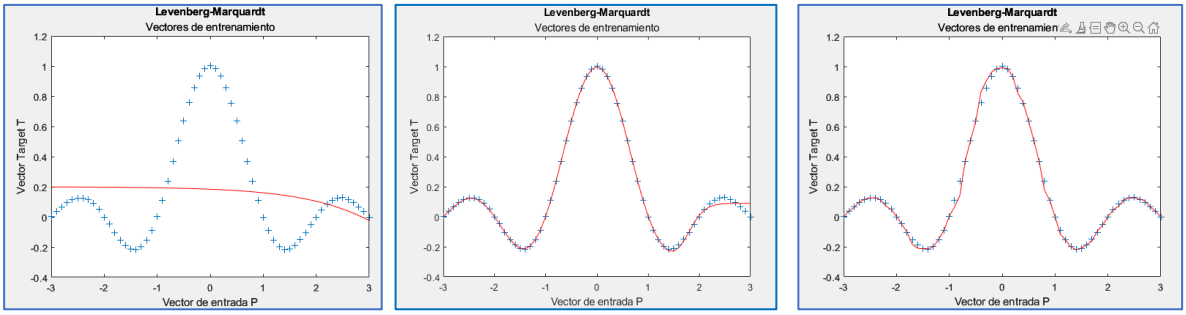


2. Levenberg-Marquardt

El parámetro 'trainlm' permite a la función *fitnet* formar una nueva red neuronal que siga este algoritmo de entrenamiento:

```
%% TRAINLM
% ==================
hiddenLayerSize = 5; % número de neuronas en la capa oculta

netLM = fitnet(hiddenLayerSize,'trainlm');

%Dividir los datos
netLM.divideParam.trainRatio = 70/100;
netLM.divideParam.valRatio = 15/100;
netLM.divideParam.testRatio = 15/100;
netLM = train(netLM,t,F);
YLM=netLM(t);

figure('Name','TRAINLM','NumberTitle','off') %Titulo de la ventana
plot(t,F,'+'); hold on;
plot(t,YLM,'-r'); hold off;
title('Levenberg-Marquardt');
subtitle('Vectores de entrenamiento');
xlabel('Vector de entrada P');
ylabel('Vector Target T');
```

Para este método, se obtienen los siguientes resultados:



- **1 neurona:** Al igual que con el método de RB, utilizar una neurona no resulta eficaz. No obstante, se observa que el carácter de la red es distinto al caso anterior, puesto que muestra una predicción no lineal.
- **5 neuronas:** El ajuste es prácticamente total, si bien con una ligera desviación en el último máximo local de la función.
- **25 neuronas:** La predicción resulta más deliberada que sus redes homólogas, aunque produce más errores que en el caso de 5 neuronas. Aún así, estos errores son más leves que el desvío de puntos de inflexión.

En esta última configuración, la red neuronal muestra la siguiente información sobre su entrenamiento:

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	9	1000
Elapsed Time	-	00:00:00	-
Performance	7.98	1.6e-06	0
Gradient	10.1	0.00114	1e-07
Mu	0.001	1e-08	1e+10
Validation Checks	0	6	6

Training Algorithms

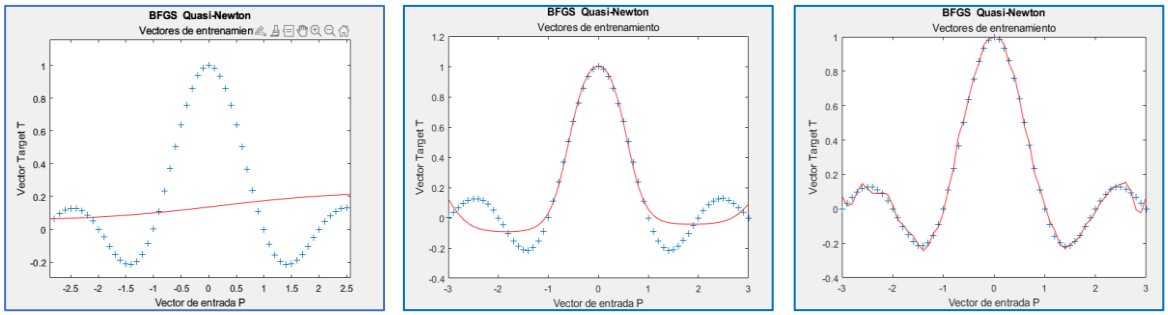
Data Division:	Random	dividerand
Training:	Levenberg-Marquardt	trainlm
Performance:	Mean Squared Error	mse
Calculations:	MEX	

3. BFGS Quasi-Newton

El parámetro 'trainbfg' permite a la función *fitnet* formar una nueva red neuronal que siga este algoritmo de entrenamiento:

```
%% TRAINBFG
% ==================
hiddenLayerSize = 5; % número de neuronas en la capa oculta

netBFG = fitnet(hiddenLayerSize,'trainbfg');

%Dividir los datos
netBFG.divideParam.trainRatio = 70/100;
netBFG.divideParam.valRatio = 15/100;
netBFG.divideParam.testRatio = 15/100;
netBFG = train(netBFG,t,F);
YBFG=netBFG(t);

figure('Name','TRAINBFG','NumberTitle','off') %Titulo de la ventana
plot(t,F,'+'); hold on;
plot(t,YBFG,'-r'); hold off;
title('BFGS  Quasi-Newton');
subtitle('Vectores de entrenamiento');
xlabel('Vector de entrada P');
ylabel('Vector Target T');
```

Para este método, se obtienen los siguientes resultados:



- **1 neurona:** Aunque se percibe un grado de ajuste algo menor con respecto a sus otras contrapartes mononeuronales, se puede observar que la predicción "acelera" y "decelera" a medida que progresa sobre la función.
- **5 neuronas:** En este caso, 5 neuronas no parecen concebir una predicción que se ajuste correctamente a los valores objetivo. El aspecto más interesante de esta red es su amplitud mostrada en los puntos de inflexión de la función.
- **25 neuronas:** La predicción resulta más deliberada que sus redes homólogas, aunque produce más errores que en el caso de 5 neuronas. Aún así, estos errores son más leves que el desvío de puntos de inflexión.

En esta última configuración, la red neuronal muestra la siguiente información sobre su entrenamiento:

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	18	1000
Elapsed Time	-	00:00:00	-
Performance	3.28	0.000676	0
Gradient	7.06	0.00401	1e-06
Validation Checks	0	6	6
Step Size	100	2	1e-06
Resets	0	0	4

Training Algorithms

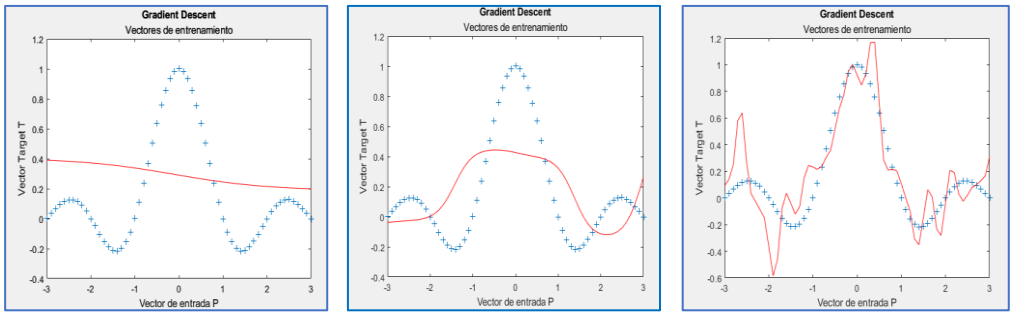
Data Division:	Random	dividerand
Training:	BFGS Quasi-Newton	trainbfg
Performance:	Mean Squared Error	mse
Calculations:	MEX	

4. Gradient Descent

Por último, el parámetro 'traingd' permite a la función *fitnet* formar una nueva red neuronal que siga este algoritmo de entrenamiento:

```
%% TRAINGD
% ==================
hiddenLayerSize = 5; % número de neuronas en la capa oculta

netGD = fitnet(hiddenLayerSize,'traingd');

%Dividir los datos
netGD.divideParam.trainRatio = 70/100;
netGD.divideParam.valRatio = 15/100;
netGD.divideParam.testRatio = 15/100;
netGD = train(netGD,t,F);
YGD=netGD(t);

figure('Name','TRAINGD','NumberTitle','off') %Titulo de la ventana
plot(t,F,'+'); hold on;
plot(t,YGD,'-r'); hold off;
title('Gradient Descent');
subtitle('Vectores de entrenamiento');
xlabel('Vector de entrada P');
ylabel('Vector Target T');
```

Para este método, se obtienen los siguientes resultados:



- **1 neurona:** Al igual que con el BFG, se puede observar que la predicción "acelera" y "decelera" a medida que progresa sobre la función.
- **5 neuronas:** La predicción resultante de esta red es la de menor calidad producida entre los distintos métodos de entrenamiento.
- **25 neuronas:** El aumento de 20 neuronas parece sobrepasar la cantidad óptima para este método en concreto, dando lugar a grandes picos cuando se alcanzan valores de validación.

En esta última configuración, la red neuronal muestra la siguiente información sobre su entrenamiento:

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	17	1000
Elapsed Time	-	00:00:00	-
Performance	1.25	0.74	0
Gradient	2.66	1.07	1e-05
Validation Checks	0	6	6

Training Algorithms

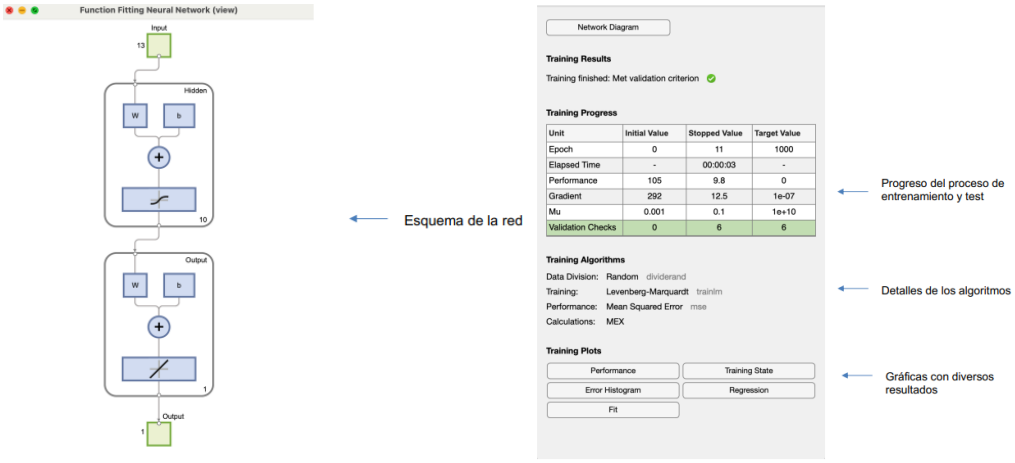
Data Division:	Random	dividerand
Training:	Gradient Descent	traingd
Performance:	Mean Squared Error	mse
Calculations:	MEX	

Ejercicio 3. Aproximación de funciones (II)

En este ejercicio, se estudiarán en detalle las herramientas que facilita Matlab para el diseño y prueba de redes neuronales ejecutando el siguiente código de ejemplo:

```
% Carga de datos de ejemplo disponibles en la toolbox
[inputs,targets] = simplefit_dataset;

% Creación de la red
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize);

% División del conjunto de datos para entrenamiento, validación y test
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Entrenamiento de la red
[net,tr] = train(net,inputs,targets);

% Prueba
outputs = net(inputs);
errors = gsubtract(outputs,targets);
performance = perform(net,targets,outputs)

% Visualización de la red
view(net)
```

Al ejecutar el script, se muestran las siguientes ventanas:

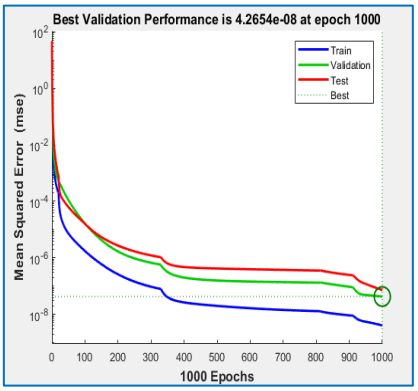


Pruebe este mismo script con el conjunto de datos bodyfat_dataset, y evalúe sus resultados. Estudie la mejora que supone utilizar distintos métodos de entrenamiento y una división diferente de los datos (entrenamiento, validación y test).

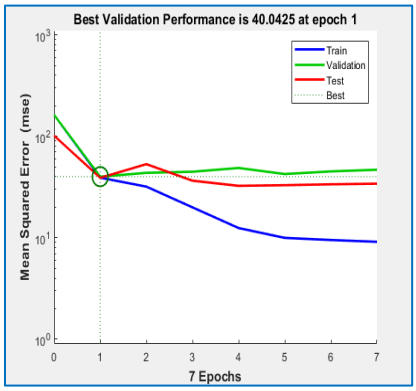
Explore las gráficas disponibles:

• **Performance:** gráfica que representa el error en función del número de épocas para los datos de entrenamiento, validación y test. Muestra también el punto en el que se producen 6 errores de ajuste en sucesivas épocas. Esta gráfica se puede utilizar para monitorizar el proceso de aprendizaje de la red.

Simplefit



Bodyfat



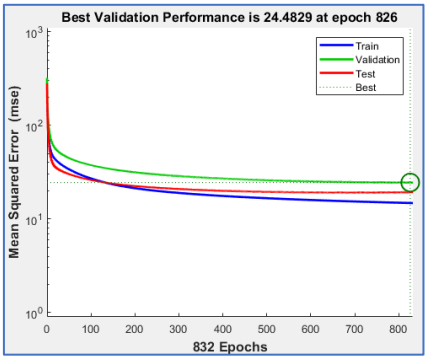
En el caso del dataset de bodyfat, el número de épocas alcanzadas antes de desviar la curva de validación respecto a la de entrenamiento es prácticamente instantáneo. Para esta red, los parámetros han sido:

- Train: 70/100
- Test: 15/100
- Validation: 15/100
- Método de entrenamiento: Resilient Backpropagation

El eje de ordenadas del gráfico denota el error cometido sobre los conjuntos de datos, luego que estas curvas no decrementen su valor indica que la red no está "aprendiendo" adecuadamente.

Sin alterar el método, se cambia la división de los campos de entrenamiento, test y validación a 60-20-20. Esto parece aumentar ligeramente el número de épocas consideradas. Aunque este número no es realmente relevante, es interesante dar cierta holgura a la selección de una época superior, puesto que el histograma de error (mostrado en su correspondiente apartado) informa de un elevado número de errores de entrenamiento, y las curvas de regresión un desajuste considerable.

Al sustituir el método de entrenamiento al BFG con 10 neuronas en la capa oculta, la red que utiliza bodyfat_dataset se aproxima a través de las épocas a su versión "óptima".



Se puede comprobar la efectividad de la nueva distribución con el error cuadrático medio de la red:

```
performanceBF =

   31.5610
```

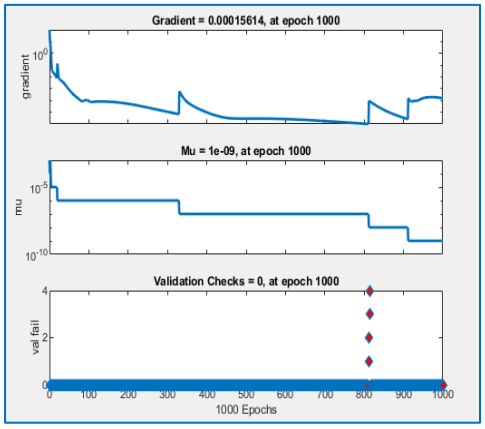
```
performanceBF =

   19.5783
```

• **Traininig State:** Representa la evolución del entrenamiento de la red neuronal a través de tres gráficas:

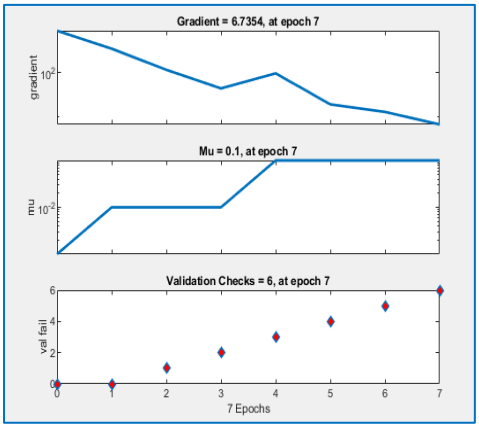
- **Gradient:** Variación del coeficiente del gradiente a través de las épocas.
- **Val fail:** Número de fallos
- **Mu:** El valor del parámetro de control del algoritmo de back-propagation.

Al cambiar de método de entrenamiento de la red a BFG, se comprueba que el parámetro "mu" desaparece, puesto que este algoritmo no lo utiliza. En su lugar, aparece una gráfica "resets", que muestra el punto en el que se alcanzan las 6 comprobaciones de validación que determinan la utilidad del aprendizaje de la red. Es un poco redundante, puesto que esta información se puede comprobar en el propio campo de "Val fail".

Simplefit

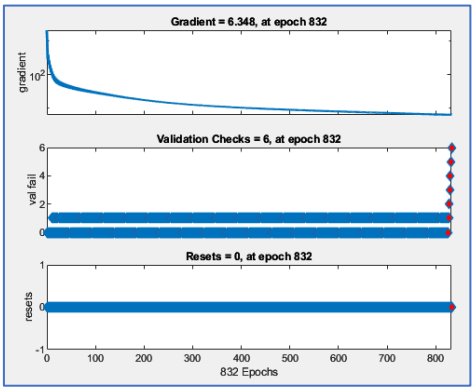


Bodyfat



En el caso de *Simplefit*, el gráfico inferior refleja el hecho de que la red no ha llegado al fallo de validación, si no al límite de épocas establecido. El gradiente sigue cambiando durante las últimas épocas, según el gráfico superior, aunque su aledaño muestra un decremento periódico del valor de control, por lo que se puede estipular que, razonablemente, el proceso de aprendizaje es finito.

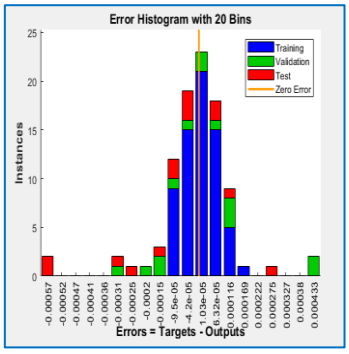
Al igual que con las gráficas de *performance*, se puede apreciar el número extremadamente reducido de épocas al que el entrenamiento ha dado lugar con el dataset de grasa corporal. Mediante BFG y la nueva distribución, la red muestra los siguientes gráficos:



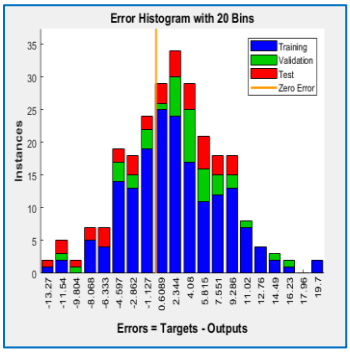
En este caso, el gradiente refleja un cambio de propagación curvo, limítrofe a una recta. Se comprueba que se alcanzan los 6 puntos de validación.

• **Error Histogram:** Estos gráficos representan la cantidad de errores que ocurre con cada uno de los datos que recibe la red, de los tres conjuntos. Se puede comprobar que la cantidad y valor de los errores en bodyfat es mucho mayor que con simplefit.

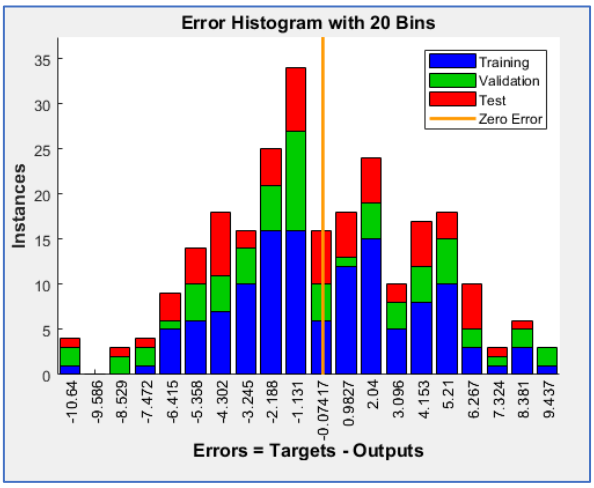
Simplefit



Bodyfat

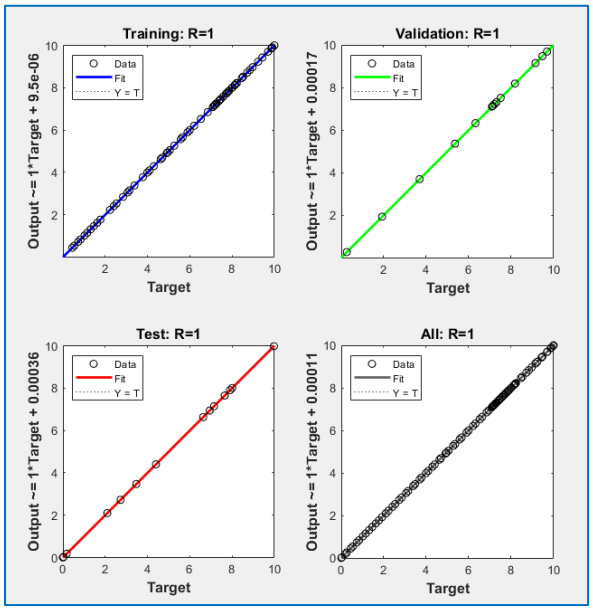


Con el algoritmo BFG y la nueva distribución de los conjuntos, el valor de los errores cometidos por la red es ligeramente menor en su distribución.

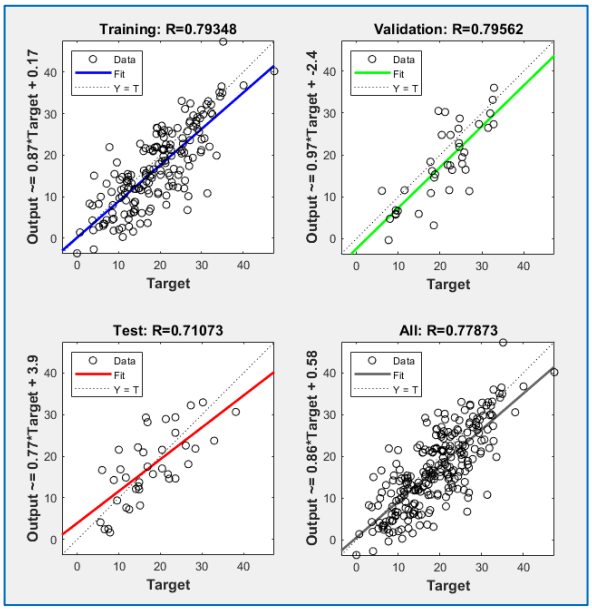


• **Regression:** Las rectas de regresión muestran el ajuste de los datos de entrenamiento, validación y test por parte de la red para cada conjunto.

Simplefit

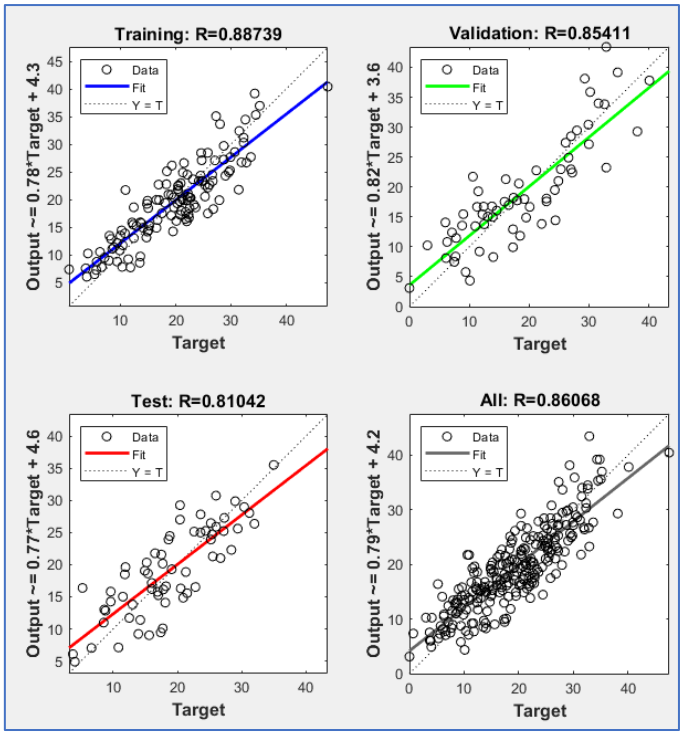


Bodyfat

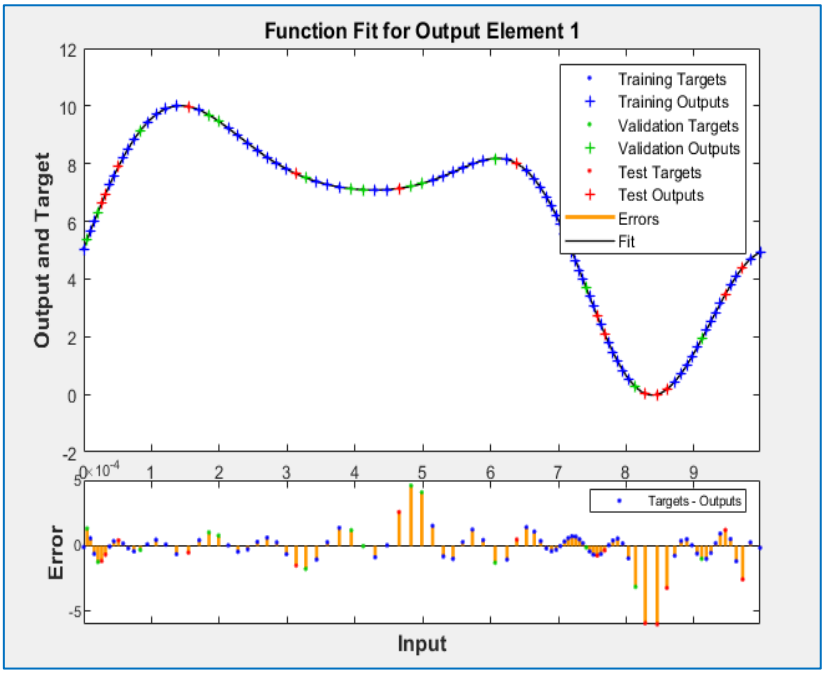


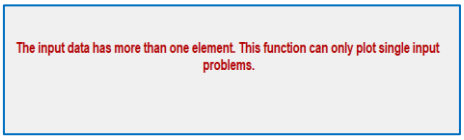
Al tratarse de un dataset más sencillo, simplefit refleja unas rectas de regresión mucho más ajustada, con todos sus valores ajustándose a las rectas obtenidas.

Por otro lado, bodyfat muestra una distribución más estocástica de los valores de sus conjuntos. Esto resulta en unas rectas más dispares respecto a las esperadas. Si se procede con la nueva configuración BFG [60,20,20], se obtienen los siguientes resultados, ligeramente más ajustados:



• **Fit:** El gráfico muestra un ajuste de la función por parte de la red. El subgráfico de error muestra la distancia entre la salida de dicha red y los valores reales esperados. Esta representación gráfica solo se puede llevar a cabo en redes con entradas unitarias, como Simplefit, lo que significa que Bodyfat, con sus 13 valores por entrada, no tiene un gráfico "fit".





Ejercicio 4. Clasificación

La clasificación de patrones es una de las aplicaciones que dieron origen a las redes neuronales artificiales. Como en el caso anterior, la toolbox de redes neuronales de Matlab dispone de una red optimizada para la clasificación, patternnet, que analizaremos en este ejemplo.

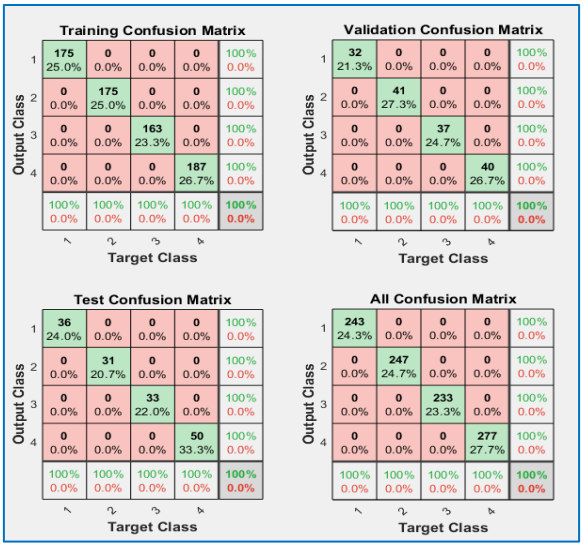
```
% Carga de datos de ejemplo disponibles en la toolbox
[inputs,targets] = simpleclass_dataset;

% Creción de una red neuronal para el reconocimiento de patrones
hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize);

% División del conjunto de datos para entrenamiento, validación y test
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Entrenamiento de la red
[net,tr] = train(net,inputs,targets);

% Prueba
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)

% Visualización
view(net)
```

La ejecución del script muestra una ventana similar a la anterior en la que cambian algunas de las gráficas disponibles para mostrar los resultados como se ve en la siguiente figura:

En lugar de las gráficas específicamente relacionadas con la aproximación de una función, en el caso de una tarea de clasificación, se ofrecen:

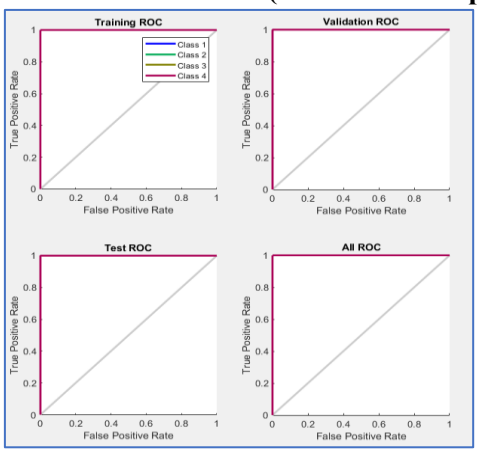
• Confusion: matrices de confusión de los resultados.

La red de este apartado utilizará el método estándar de entrenamiento y una distribución [70,15,15].



La matriz de confusión obtenida no refleja ningún falso positivo o falso negativo; de hecho, se obtiene un *true positive ratio* del 100%. Esto da a entender que el resultado obtenido es más que deseable.

• Receiver Operating Charactersitic: curvas ROC (característica operativa del receptor).



Las curvas ROC (*Receiver Operating Characteristics*) confirman que la red resulta extremadamente precisa, puesto que son completamente distantes en forma a la diagonal del 50% de verdaderos y falsos positivos. Esta métrica de acierto es de gran utilidad, puesto que se tienen en cuenta todas las posibilidades de umbral para casos particulares.

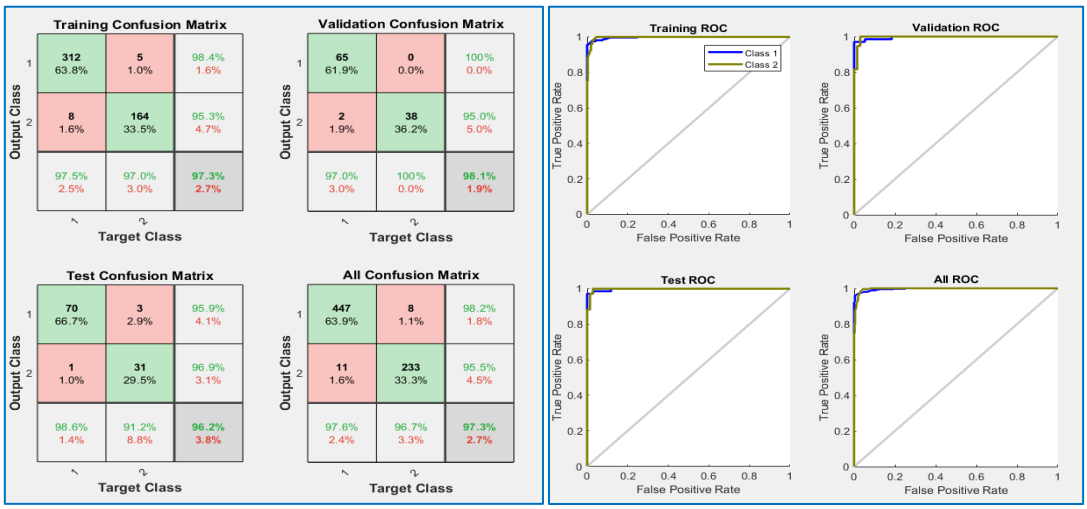
Pruebe este mismo script con el conjunto de datos cancer_dataset, y evalúe sus resultados. Estudie de nuevo la mejora que supone utilizar distintos métodos de entrenamiento y una división diferente de los datos (entrenamiento, validación y test).

1. Resilient Backpropagation

Para el método de entrenamiento trainRP (estándar) y con una división de datos de:

```
% División del conjunto de datos para entrenamiento, validación y test
netSC.divideParam.trainRatio = 70/100;
netSC.divideParam.valRatio = 15/100;
netSC.divideParam.testRatio = 15/100;
```

Se obtienen:

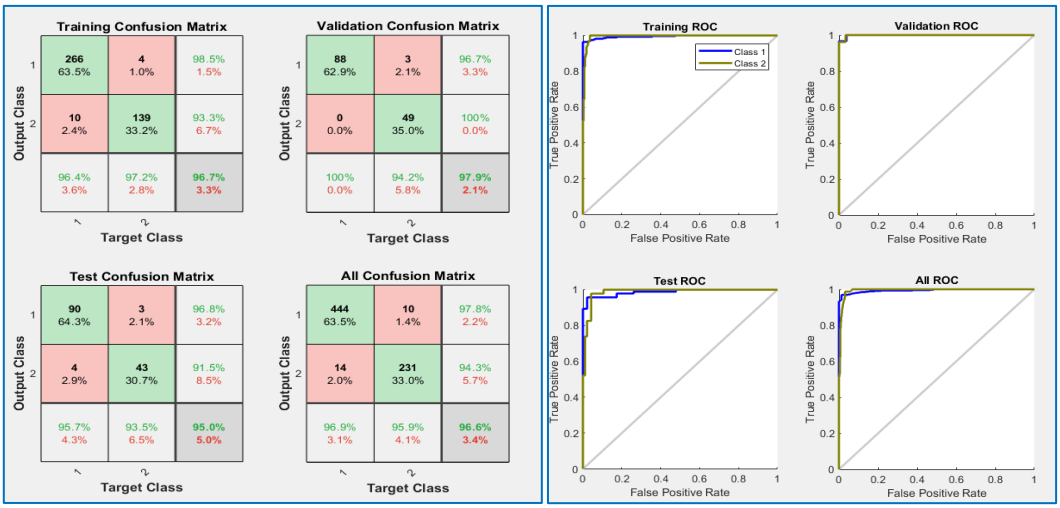


En este caso, la proporción de aciertos positivos sigue siendo bastante alta, por encima del 95% en todos los casos. Las curvas ROC reflejan cierta concavidad, aunque siguen siendo muy distantes a la forma de la diagonal.

Cambiando la división de datos a la siguiente:

```
% División del conjunto de datos para entrenamiento, validación y test
netC.divideParam.trainRatio = 60/100;
netC.divideParam.valRatio = 20/100;
netC.divideParam.testRatio = 20/100;
```

Se obtienen:



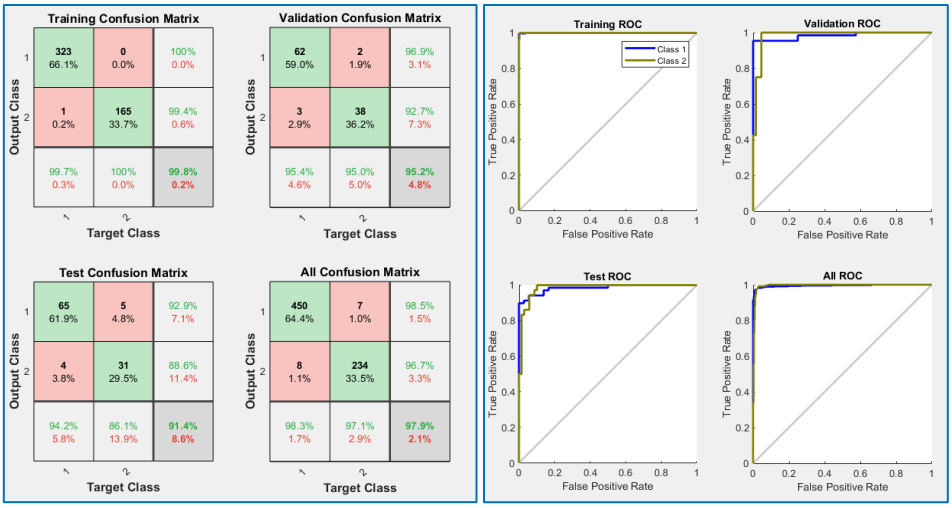
Con esta distribución, las proporciones resultan ligeramente menos deseables, habiendo encontrado más falsas clasificaciones notablemente en el conjunto de test.

2. Levenberg-Marquardt

Para el método de entrenamiento trainLM, con el mismo número de neuronas en la capa oculta, y con una división de datos de:

```
% División del conjunto de datos para entrenamiento, validación y test
netC.divideParam.trainRatio = 70/100;
netC.divideParam.valRatio = 15/100;
netC.divideParam.testRatio = 15/100;
```

Se obtiene:

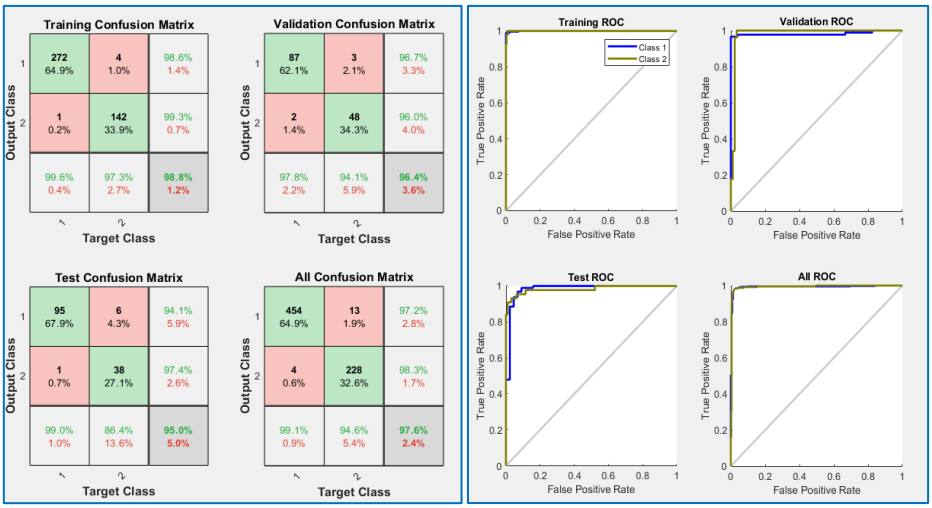


Se obtienen resultados incluso peores que con el algoritmo de aprendizaje anterior.

Cambiando la división de datos a la siguiente:

```
% División del conjunto de datos para entrenamiento, validación y test
netC.divideParam.trainRatio = 60/100;
netC.divideParam.valRatio = 20/100;
netC.divideParam.testRatio = 20/100;
```

Se obtienen:

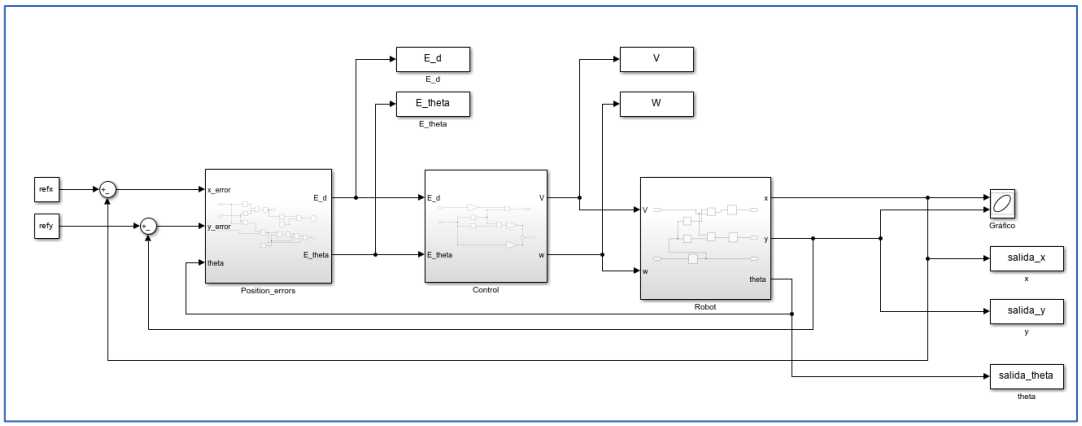


En este caso, una distribución con un menor tamaño de entrenamiento parece dar mejores resultados.

PARTE 2

Diseño de un control de posición mediante una red neuronal no recursiva

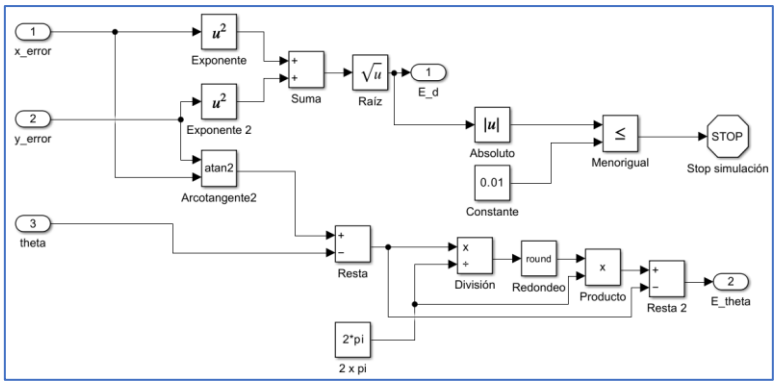
A partir del robot creado en la práctica 0 y el módulo "Control", proporcionado para esta segunda parte, se crea un sistema nuevo.



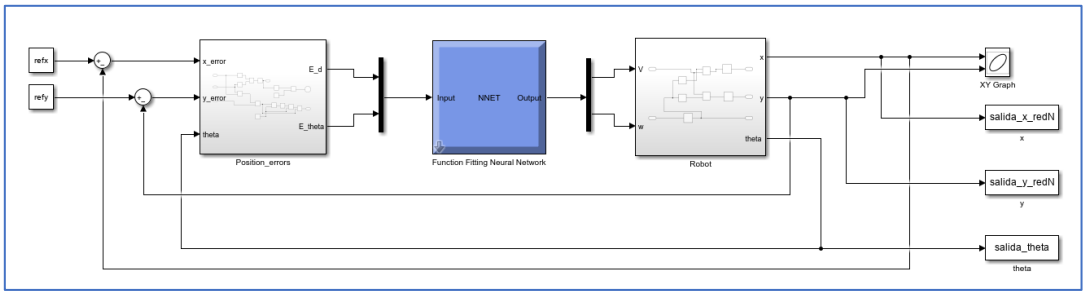
Como nuevo componente, se crea un subsistema "Position_errors", encargado de controlar los errores de posición y de ángulo al objetivo utilizados para el control de posición del robot.

- Error de distancia al objetivo: $E_d = sqrt((refx - x_k)^2 + (refy - y_k)^2)$
- Error de ángulo al objetivo: $E_\theta = atan2(refy - y_k, refx - x_k) - \theta_k$

Además, asegura que los ángulos proporcionados se mantengan en el rango [-π,π], puesto que la dirección del robot se maneja en ese dominio. Por último, permite que se detenga el robot si se encuentra a una distancia menor a 0.01 unidades del punto objetivo.



Tras su implementación, se conecta el modelo "PL1_P2_PositionControl.slx" al archivo de código "PL1_P2_RunPositionControl.m", desde el cual se crea un dataset a partir de los datos obtenidos del funcionamiento del sistema con la "caja negra".



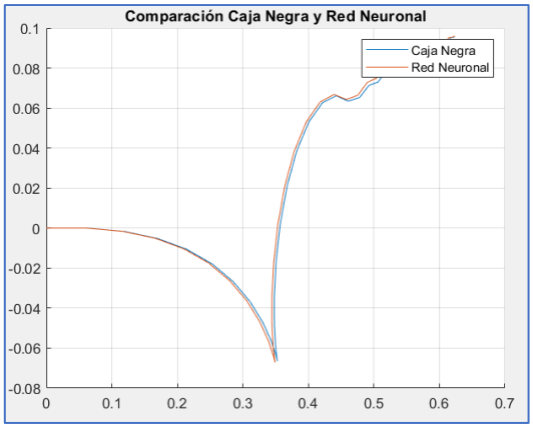
Dicho dataset se genera a partir de los siguientes parámetros:

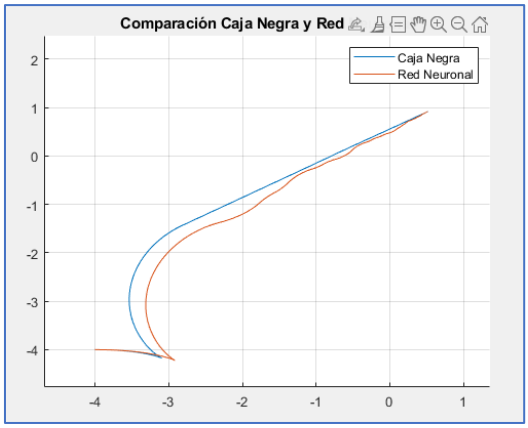
- Valores refx y refy, pertenecientes al rango [-10, 10].
- División de conjuntos:
  - 80% entrenamiento
  - 10% validación
  - 10% test

  Esta división, al igual que el número de neuronas en la capa oculta, ha sido obtenida de forma empírica.

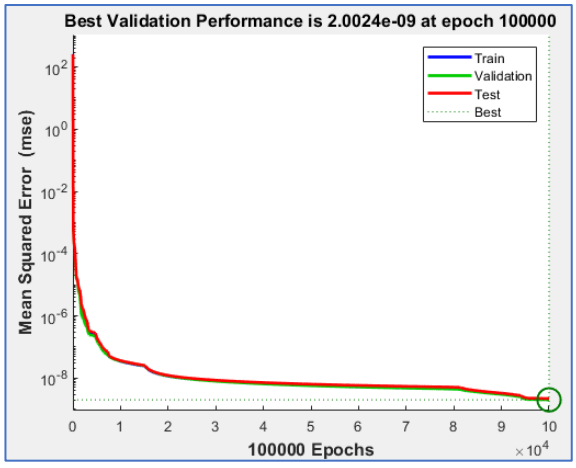
- Número de neuronas en la capa oculta: 12

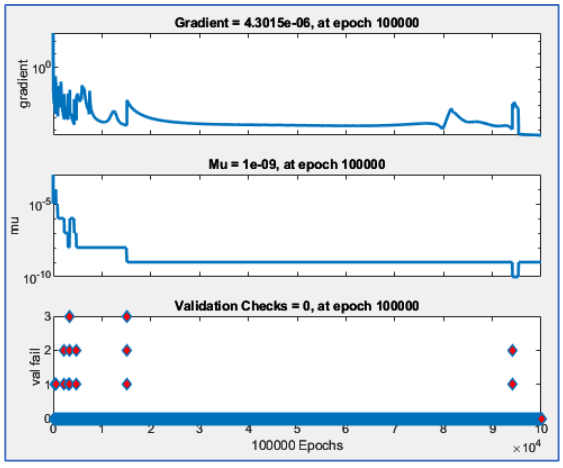
La red resultante es capaz de predecir la función en su totalidad sin error, e incluso puede alcanzar el mismo punto final, partiendo desde un origen distinto, siguiendo una trayectoria similar a la del robot original:



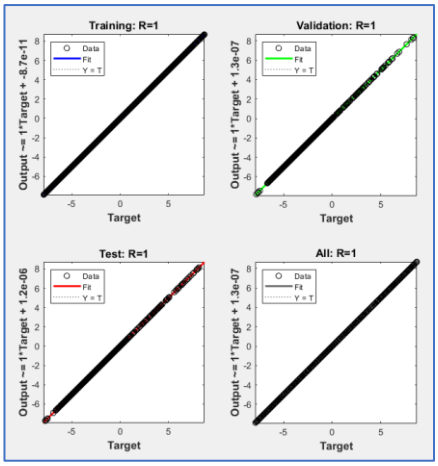


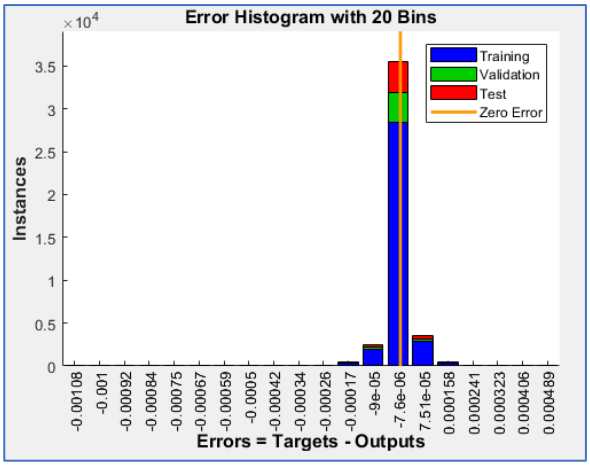
Para obtener esta red, se han recorrido 100.000 épocas, como reflejan los gráficos:





Las curvas de regresión y el histograma de errores muestran que el ajuste es prácticamente absoluto, tanto en entrenamiento y validación como en test:





Los parámetros del progreso de entrenamiento reflejan un buen rendimiento de la red, con un error cuadrático medio ínfimo:

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	100000	100000
Elapsed Time	-	00:54:43	-
Performance	259	2.04e-09	0
Gradient	446	4.3e-06	1e-07
Mu	0.001	1e-09	1e+10
Validation Checks	0	0	6

Training Algorithms

Data Division:	Random	dividerand
Training:	Levenberg-Marquardt	trainlm
Performance:	Mean Squared Error	mse
Calculations:	MEX	

Se concluye, pues, que el comportamiento de la red es similar al de la caja negra.