

PRÁCTICA 1 - IDENTIFICACIÓN Y CONTROL NEURONAL

**Mario Fernández Rueda
Francisco González Velasco
PUESTO 10 DE LABORATORIO DEL
GRUPO 4A2**



Primera Parte

Ejercicio 1

Se realiza lo especificado en el enunciado con el siguiente código:

```
% Datos de entrada (inputs)
X = [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];
% Clases a las que pertenecen los datos
% en binario para representar:
% 0, 0 = 0; 0, 1 = 1; 1, 0 = 2; 1, 1 = 3
Y = [1 1 1 0 0 1 1 1 0 0;
     0 0 0 0 0 1 1 1 1 1];

figure;
subplot(2, 2, 1); %subplot de 2 filas x 2 columnas. ocupa la posición 1
plotpv(X, Y);

% Crear una red neuronal
net = perceptron;
% Entrenar la red neuronal
net = train(net, X, Y);

subplot(2, 2, 2); %subplot de 2 filas x 2 columnas. ocupa la posición 1
plotpv(X, Y);
plotpc(net.iw{1, 1}, net.b{1});

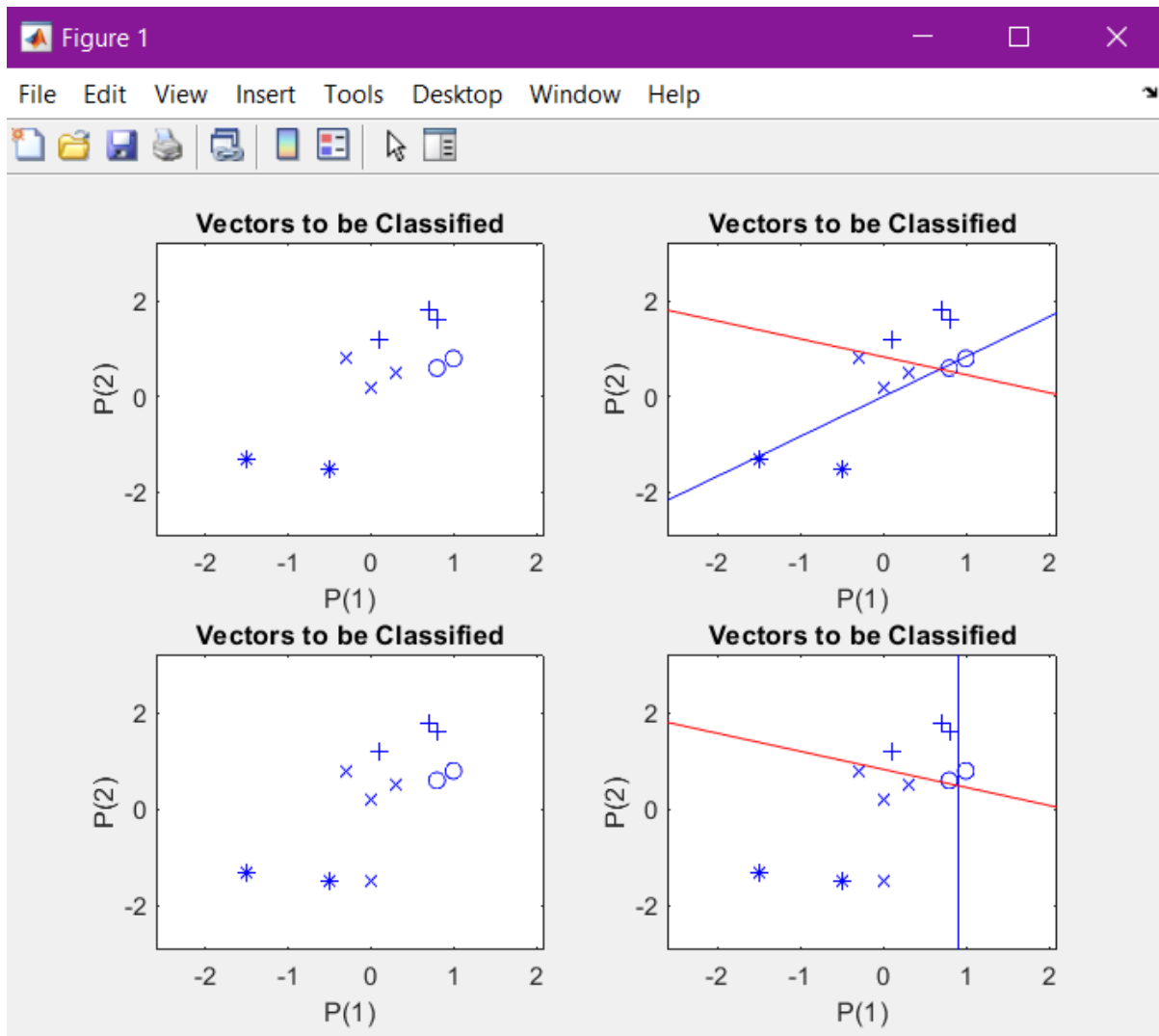
% Agregar el nuevo dato [0.0 -1.5]
nuevo_dato = [0.0; -1.5];
X = [X nuevo_dato];
Y = [Y [1; 1]];

% Visualizar los datos actualizados
subplot(2, 2, 3);
plotpv(X, Y);

% Se vuelve a entrenar la red neuronal
net = train(net, X, Y);

% Visualizar la superficie de separación
subplot(2, 2, 4);
plotpv(X, Y);
plotpc(net.iw{1, 1}, net.b{1});
```

Usamos la clase perceptrón de Matlab y con train entrenamos al perceptrón, obteniendo así 2 pesos, debido a los 2 elementos del input y un sesgo.



En primer lugar, se muestran los puntos de las diferentes clases, sin obtener el modelo, en la segunda imagen se ve el modelo creado por el perceptrón que separa de manera correcta las diferentes clases con 2 líneas para obtener una división de 4 clases. En la imagen de abajo a la izquierda se muestran los datos con el nuevo introducido. Se ve que va en una zona donde no le corresponde a su clase con el anterior modelo. También se aprecia que el problema se convierte en uno linealmente no separable por rectas como hace el perceptrón, haciendo que, de ninguna manera usando este tipo de modelo se pueda obtener una correcta clasificación. Igualmente, probamos a hacerlo y evidentemente se ve que no hace una correcta clasificación. Por tanto, lo que podemos sacar en conclusión es que el perceptrón sí que consigue separar los datos con los datos iniciales, pero al introducir el nuevo dato no es capaz de crear un modelo que clasifique correctamente. Con otro tipo de modelo que usa líneas curvas sí que se podría. La capa de salida debe tener 2 neuronas; con solo una, como sugiere el enunciado, no se podría clasificar adecuadamente. Esto se debe a que un perceptrón solo puede distinguir entre dos valores con su función hardlim. Con 2 neuronas, la red puede clasificar entre los 4 valores necesarios.

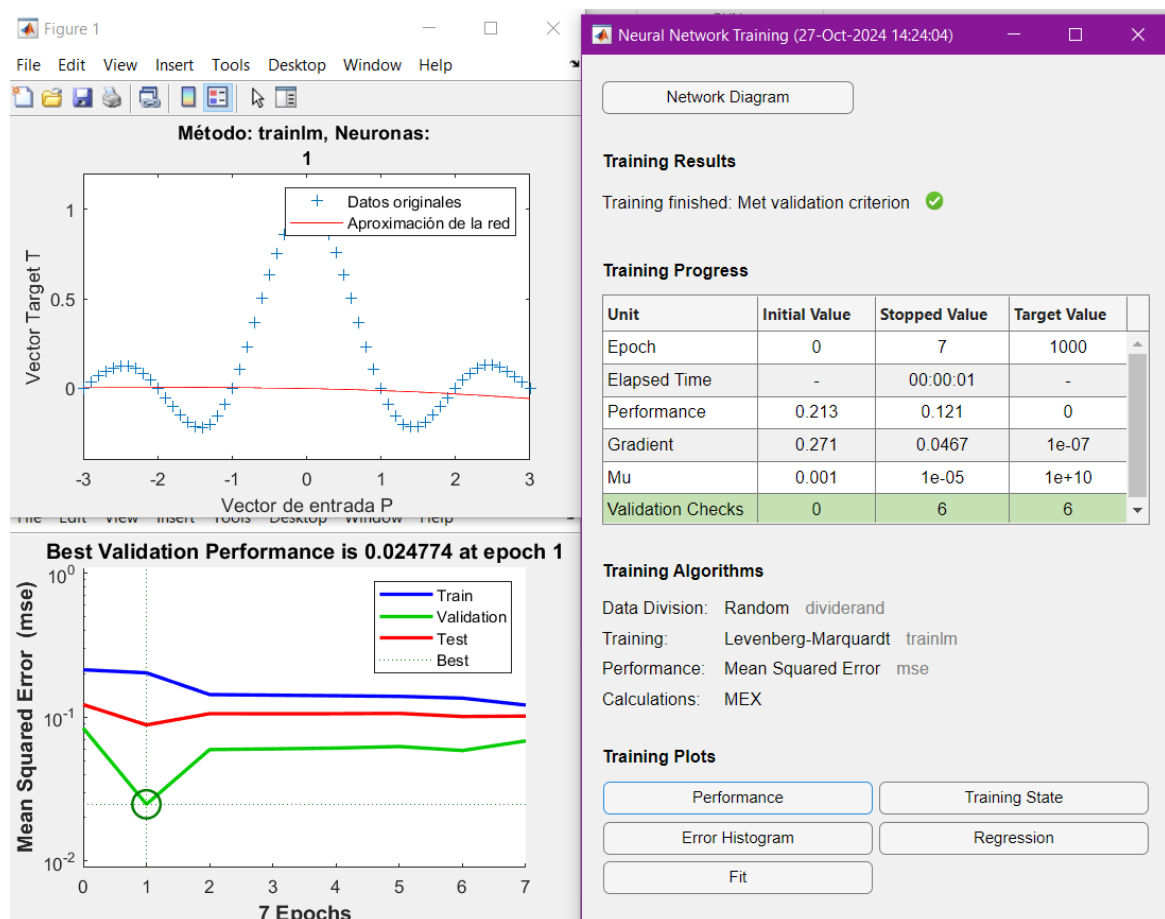
Ejercicio 2

Probamos distintos números de neuronas de la capa oculta de la red: **1, 4, 10, 15, 20**

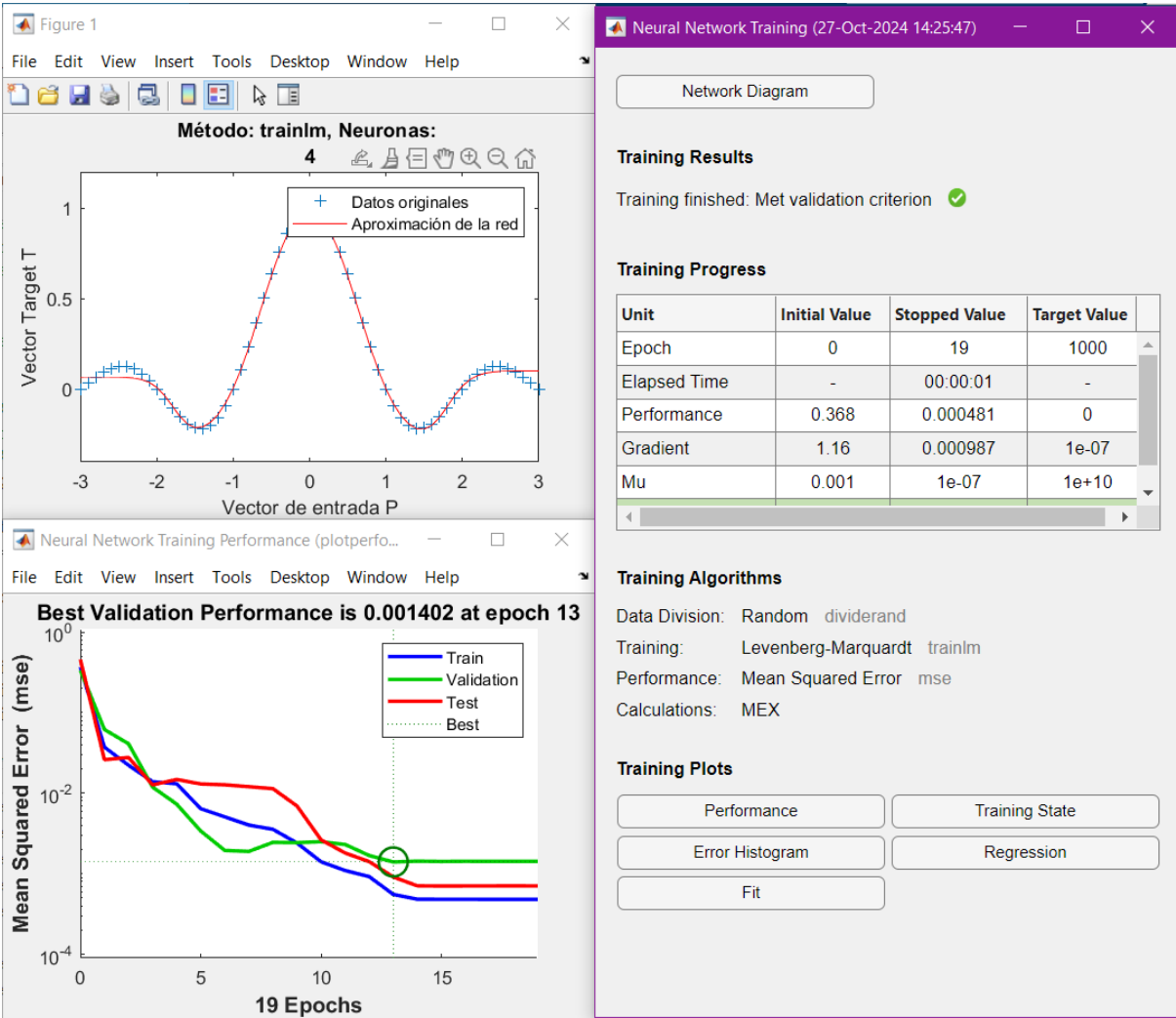
Con 1, 4, 10, 15 y 20 neuronas buscamos distintos niveles de ajuste. **1 neurona** para ver lo que hace con la mínima cantidad posible. Con **4 y 10 neuronas** se busca buena precisión y detalles sin caer en sobreajuste. **15 neuronas**, empieza a ser un número alto, debería ajustar más, aunque empiezan a mostrar signos de sobreajuste, y con **20 neuronas**, un número ya alto, la red se adaptaría demasiado a los datos, perdiendo generalización.

Se probarán con el método de entrenamiento por defecto 'trainlm'.

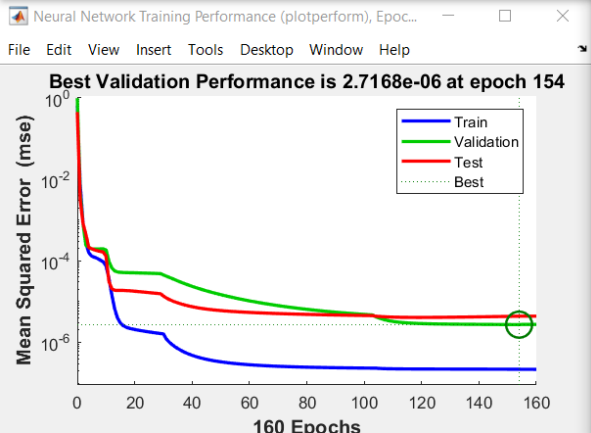
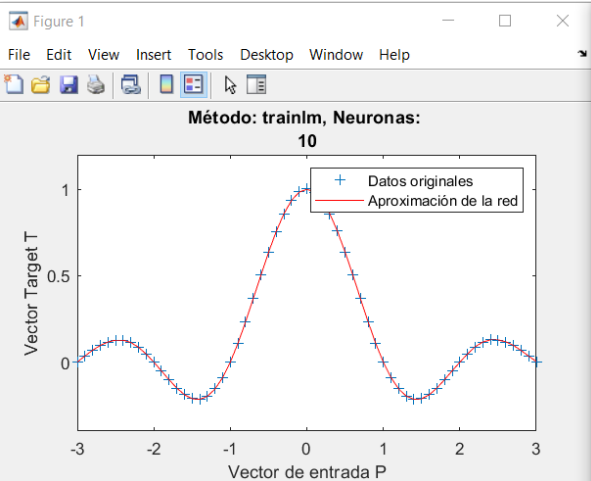
1 neurona:



4 neuronas:



10 neuronas:



Neural Network Training (27-Oct-2024 14:26:53)

Network Diagram

Training Results

Training finished: Met validation criterion ✓

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	160	1000
Elapsed Time	-	00:00:01	-
Performance	0.64	2.18e-07	0
Gradient	1.3	5.56e-07	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

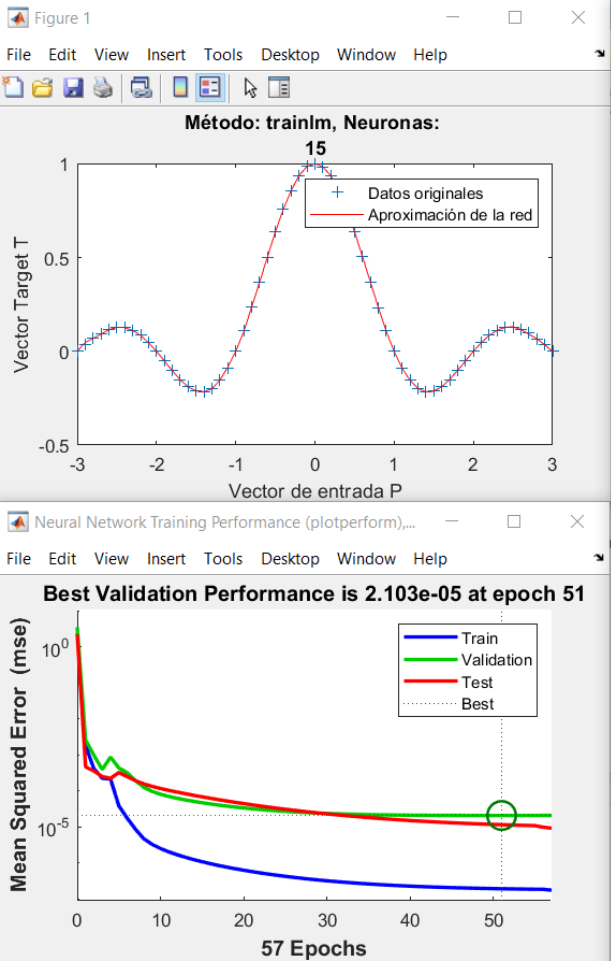
Training Plots

Performance Training State

Error Histogram Regression

Fit

15 neuronas:



Neural Network Training (27-Oct-2024 14:27:54)

Network Diagram

Training Results

Training finished: Met validation criterion ✓

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	57	1000
Elapsed Time	-	00:00:01	-
Performance	2.39	1.77e-07	0
Gradient	4.1	6.38e-05	1e-07
Mu	0.001	1e-09	1e+10
Validation Checks	0	6	6

Training Algorithms

Data Division: Random dividerand

Training: Levenberg-Marquardt trainlm

Performance: Mean Squared Error mse

Calculations: MEX

Training Plots

Performance

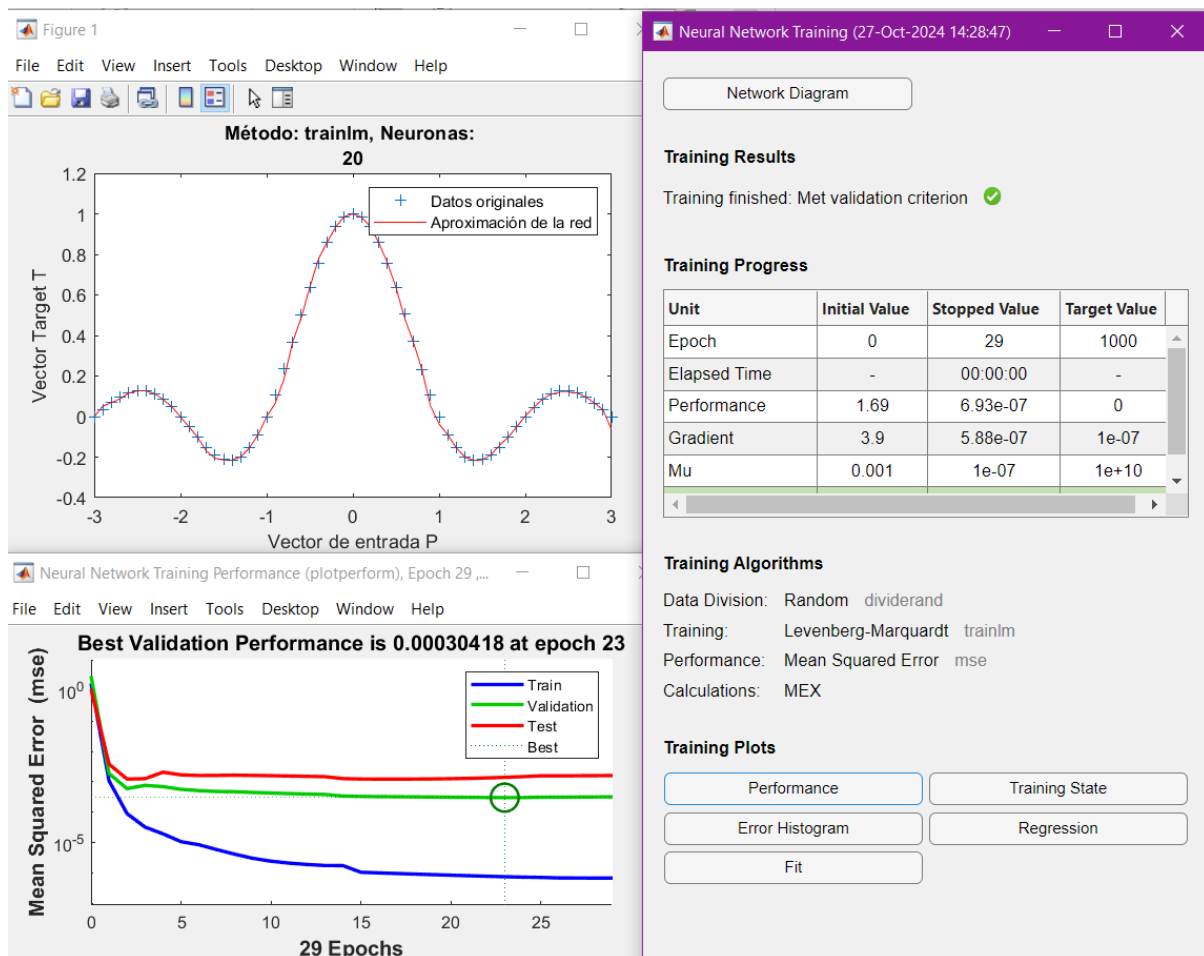
Training State

Error Histogram

Regression

Fit

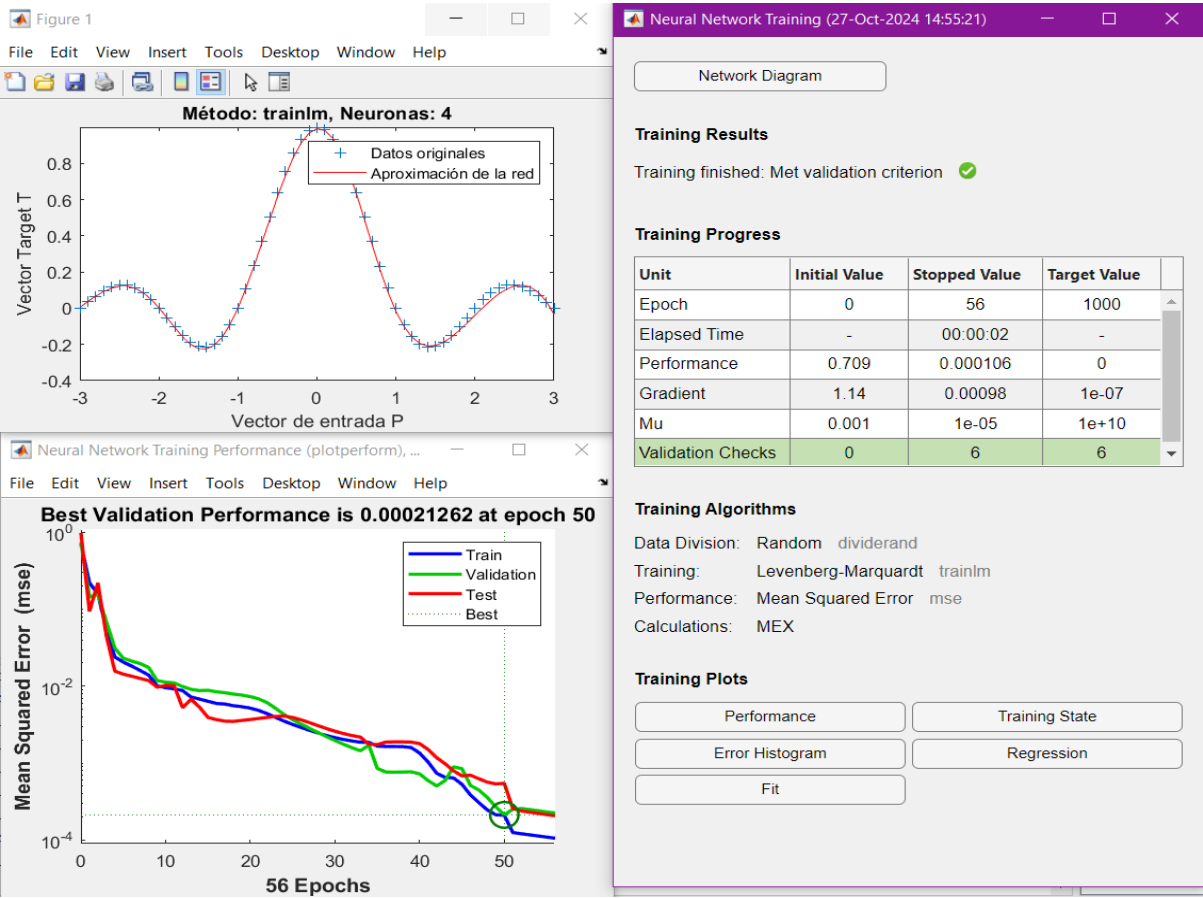
20 neuronas:



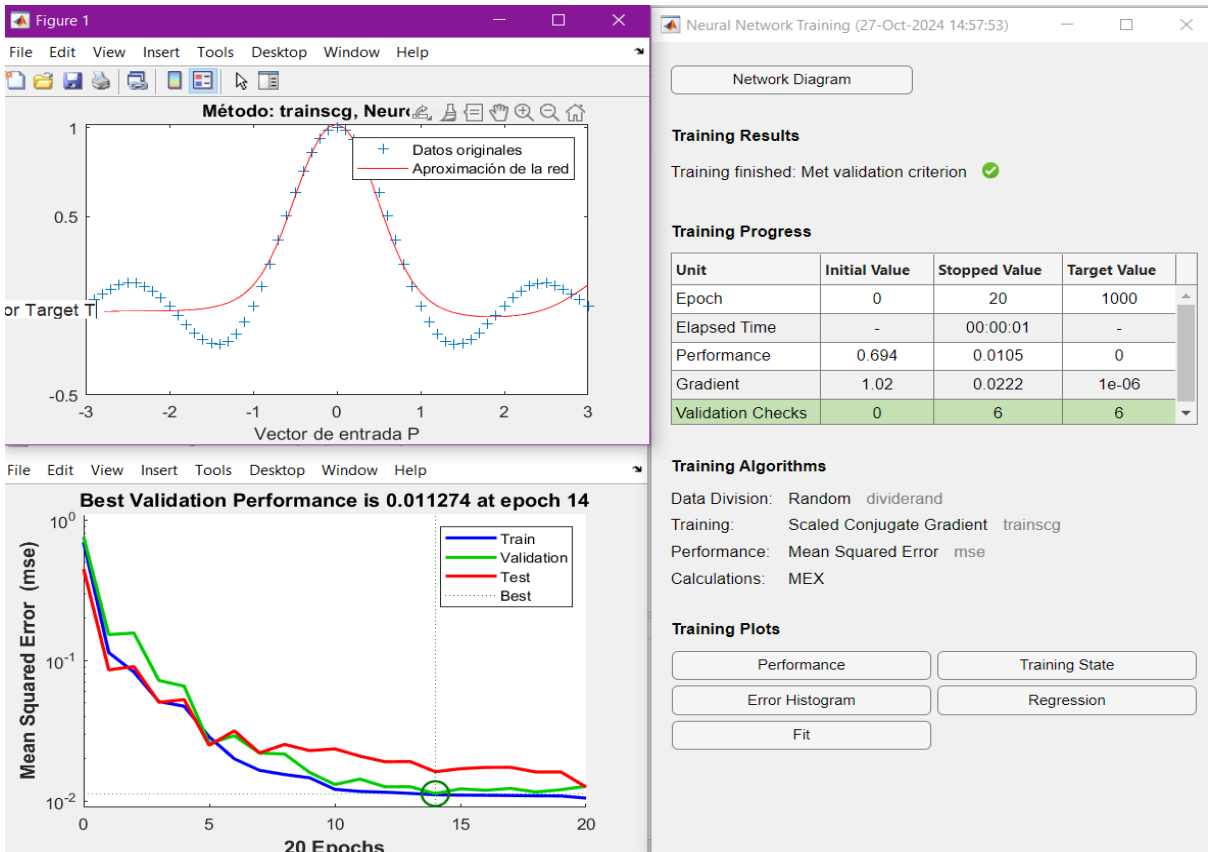
Se cumple lo previsto en el análisis del tamaño hecho previamente, siendo 4 y 10 los mejores tamaños, ya que en 15 se empieza a ver sobreajuste, debido ya al número excesivo de neuronas y 20 claramente con la línea se ve que se va a puntos concretos, siendo muy específica la aproximación. 1 evidentemente, no aproxima nada, por lo que no es útil. Lo más seguro es que 4 sea el número de neuronas que aproxima decentemente, por lo que habría que encontrar un equilibrio entre 4 y 10.

Ahora vamos con los métodos de entrenamiento seleccionados, que son trainlm, trainbr, traingd, y trainscg. Se prueban ahora con el número de neuronas 4, que es el que nos da el código del enunciado:

Método trainlm:



Método trainscg:



Neural Network Training (27-Oct-2024 14:57:53)

Network Diagram

Training Results

Training finished: Met validation criterion

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	20	1000
Elapsed Time	-	00:00:01	-
Performance	0.694	0.0105	0
Gradient	1.02	0.0222	1e-06
Validation Checks	0	6	6

Training Algorithms

Data Division: Random dividerand

Training: Scaled Conjugate Gradient trainscg

Performance: Mean Squared Error mse

Calculations: MEX

Training Plots

Performance

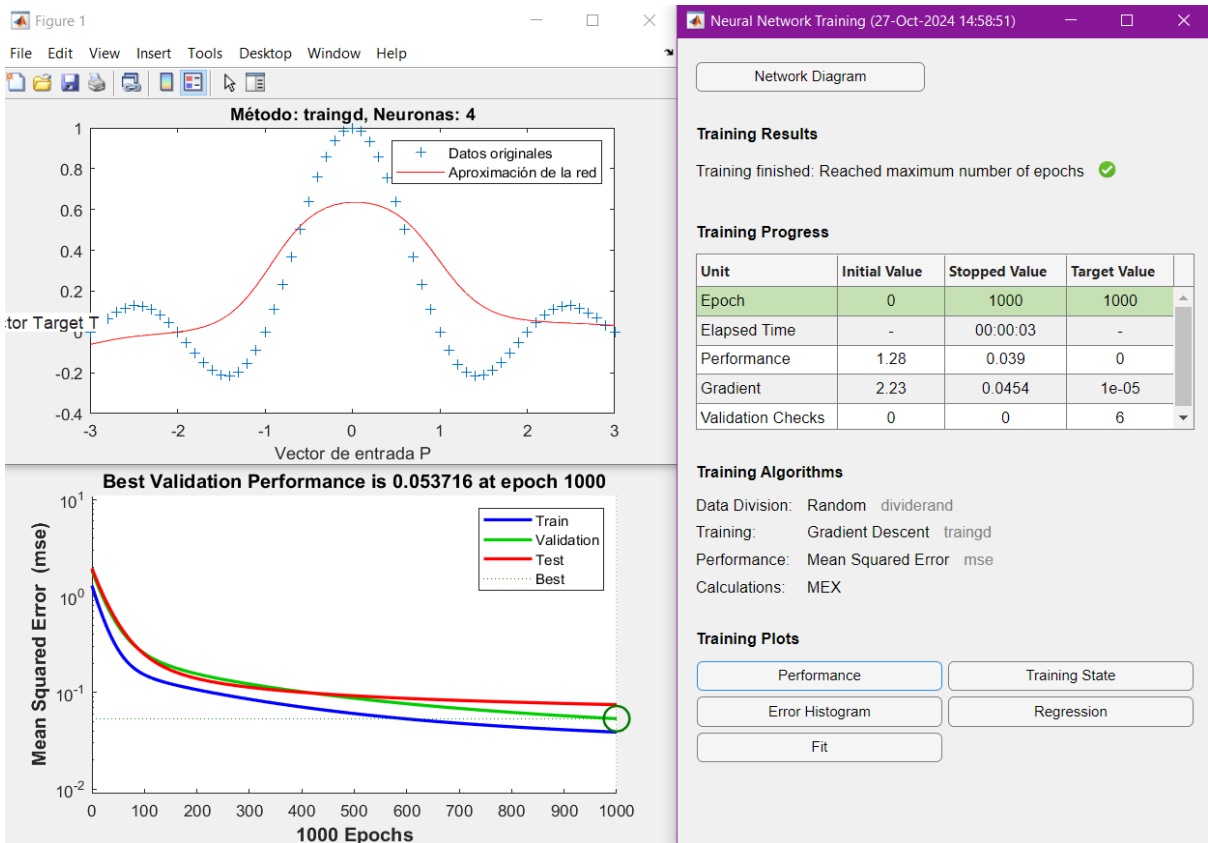
Training State

Error Histogram

Regression

Fit

Método traingd:



Neural Network Training (27-Oct-2024 14:58:51)

Network Diagram

Training Results

Training finished: Reached maximum number of epochs

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	1000	1000
Elapsed Time	-	00:00:03	-
Performance	1.28	0.039	0
Gradient	2.23	0.0454	1e-05
Validation Checks	0	0	6

Training Algorithms

Data Division: Random dividerand

Training: Gradient Descent traingd

Performance: Mean Squared Error mse

Calculations: MEX

Training Plots

Performance

Training State

Error Histogram

Regression

Fit

A partir de las gráficas, parece que el método trainbr es el mejor seguido del trainlm, que es el que es por defecto. Para esta red con este número de neuronas es ideal trainbr, ya que aproxima muy bien con el número de neuronas mínimo decente que es 4. Trainscg, aproxima bien en unas partes y traingd, no aproxima bien, siendo muy general. Para otras redes puede que sí aproxime mejor, pero no para esta.

Conclusiones:

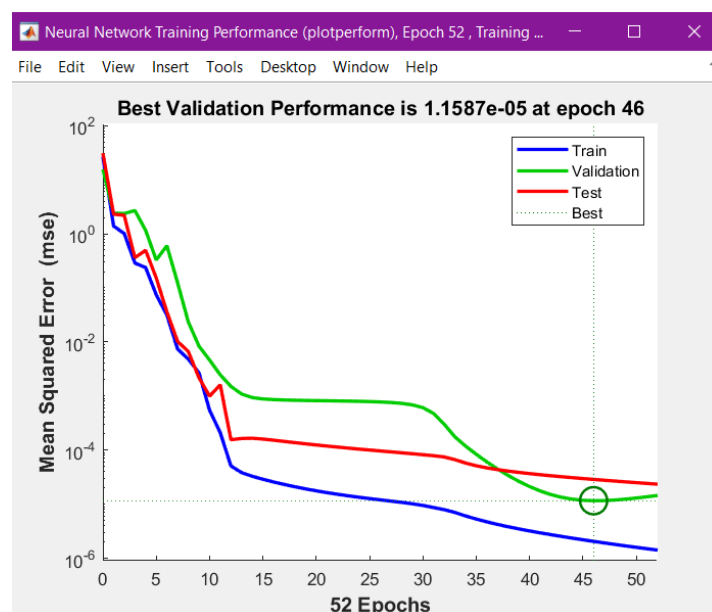
El método de entrenamiento y la cantidad de neuronas en la capa oculta son factores fundamentales para ajustar y optimizar una red neuronal. Es necesario probar y experimentar con todas las posibles de argumentos para dar con la mejor combinación para el problema específico. En este caso hemos dado con trainbr y 4-10 neuronas, pero para otro problema puede que la combinación cambie. Al igual que esta es la mejor combinación, hay otros factores que podrían cambiarse como la división de datos de entrenamiento o la función de activación, que habría que estudiar y probar para cada red.

Ejercicio 3

Analicemos las diferentes ventanas que nos da Matlab cuando usamos la función 'train' de redes neuronales. Se hace con el dataset simplefit.

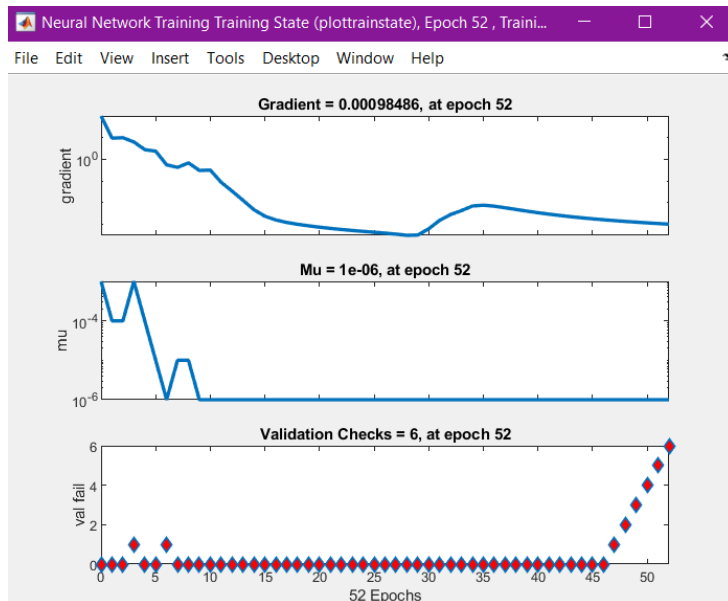
Performance:

Proporciona un gráfico de líneas que indica el error de la red en cada época, lo que ayuda a evaluar si la red está aprendiendo adecuadamente y si está ajustándose bien a los datos. Muestra el error, diferencia entre los valores predichos y los valores reales, de los datos de entrenamiento, validación y prueba y la época con mejor valor de error, así como las épocas totales que se han realizado en la red.



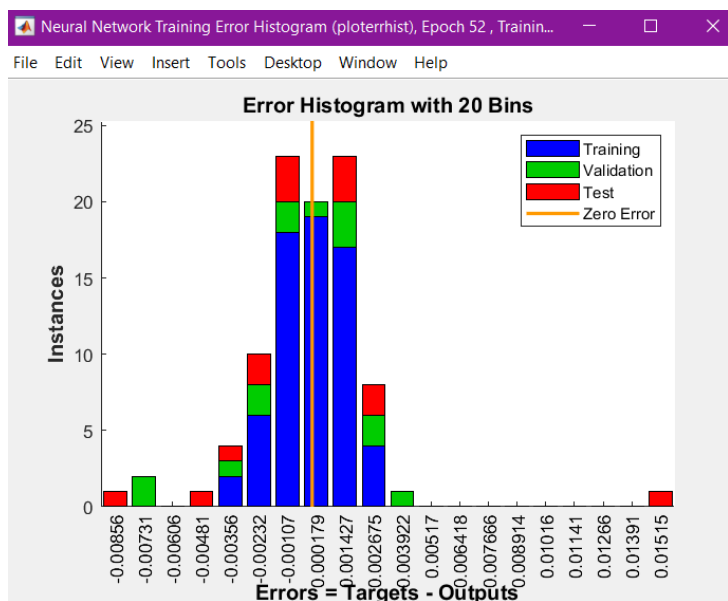
Training state:

Muestra el número de épocas completadas, el gradiente del error, la tasa de aprendizaje y el número de fallos de validación. Todo esto a lo largo de las épocas completadas.



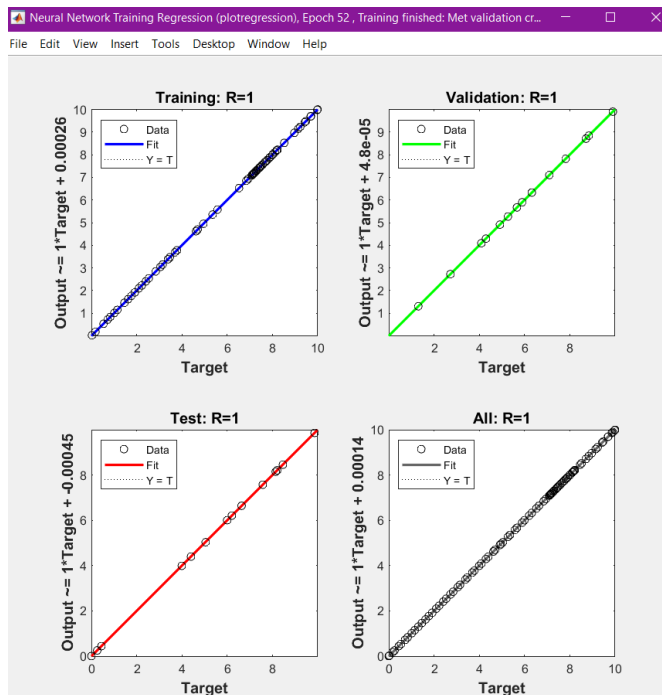
Error histogram:

Muestra un histograma de los errores de la red. En el eje x se muestra el valor del error, dividido desde el valor más bajo al más alto en 20 intervalos. En el eje y la cantidad de veces que se ha dado el valor del error, tanto del entrenamiento, validación y test. Una línea muestra el punto del eje x con valor 0.



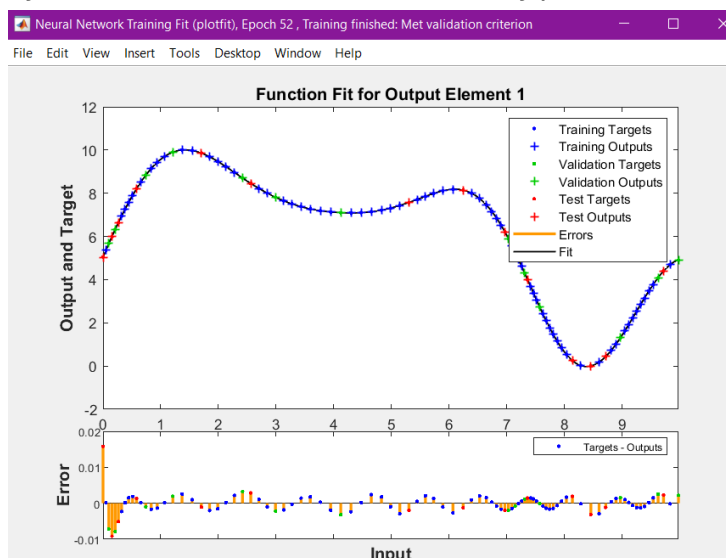
Regression:

Muestra 4 gráficas que evalúan la precisión del modelo en términos de regresión para los conjuntos de entrenamiento, validación, test y todos juntos. Es útil para visualizar la precisión del modelo y ver cómo de bien se ajusta a los datos. En este gráfico, se puede observar la línea de regresión ideal (donde las predicciones coinciden perfectamente con los valores reales) y cómo se distribuyen los puntos alrededor de esta. Aquí todos se ajustan perfectamente.

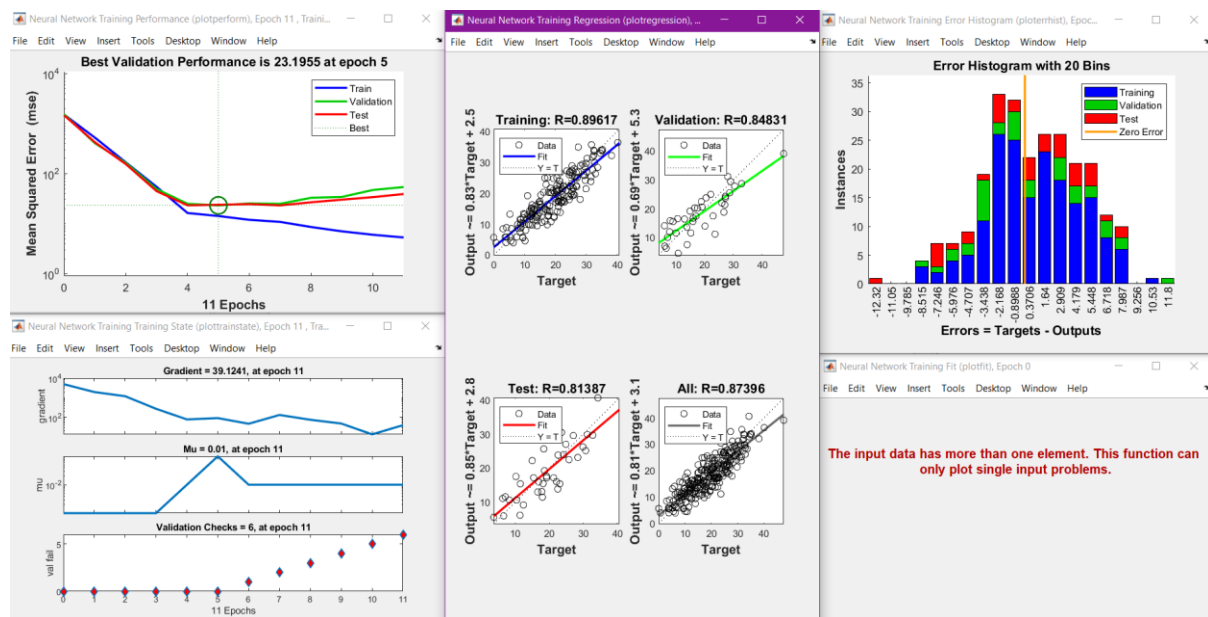


Fit:

Muestra en el gráfico de arriba cómo la función de salida de la red se ajusta en el intervalo de entradas, mostrando tanto los puntos de los valores objetivo (targets) como las predicciones de la red para esas mismas entradas. Además, las barras de error indican las diferencias entre las predicciones de la red y los valores objetivo, ayudando a visualizar el rendimiento y precisión de la red.



Con el dataset bodyfat_dataset tenemos los siguientes resultados:



Comparamos ahora las gráficas de ambos datasets, excepto el fit, que en el dataset de bodyfat no aparece debido a que los datos tienen más de un elemento de input.

Con la gráfica de performance se puede ver que el primer dataset mejora siempre, su error de test siempre baja, siendo la mejor época en cuanto al error la última. Mientras tanto la de bodyfat hace bastantes menos épocas y no es la última la mejor época, si no que después de la mejor, el error de test sube. Esto nos dice que puede que los datos de simplefit sean más sencillos de aproximar a una red que los de bodyfat. Sin embargo, en cuanto a tiempo es peor el dataset de simplefit, ya que tarda más épocas para encontrar el mejor modelo. Esto puede deberse a la función de performance elegida.

Viendo las otras gráficas también conformamos que el dataset de bodyfat tiene datos menos sencillos, que se ajustan menos a lo ideal (las líneas), encontrando algún outlayer como nos muestra la pestaña regression. A su vez encontramos más errores en la ventana Error Histogram.

Probamos ahora distintos métodos de entrenamiento sobre la dataset simplefit. Seleccionamos los tres que no son el default (trainlm) del anterior ejercicio y medimos la performance que al ser MSE (error cuadrático medio) buscamos el valor más bajo posible:

<code>trainbr</code>	<code>traingd</code>	<code>trainscg</code>
<code>performance =</code>	<code>performance =</code>	<code>performance =</code>
4.6004e-10	26.4977	0.1185

El mejor sigue siendo trainbr con el menor valor de los tres.

Ahora probamos diferentes valores para los datos de entrenamiento con la misma dataset y el mismo método de entrenamiento.

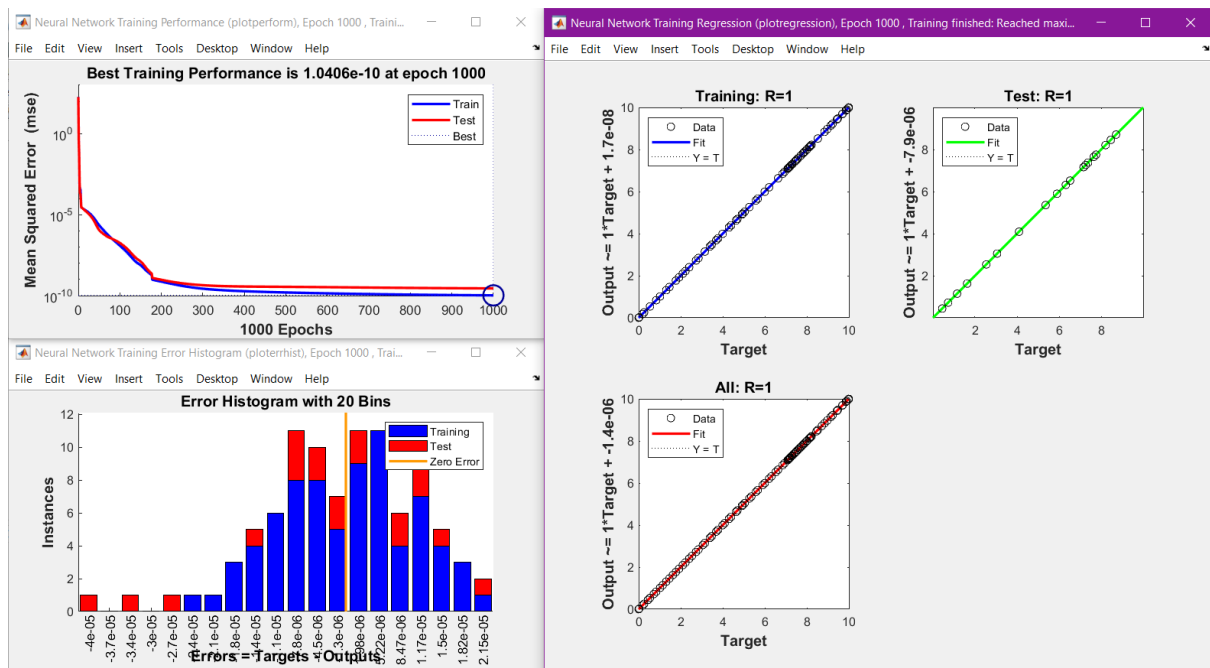
La función de entrenamiento anterior se medió con una división de datos de 70/15/15 (train, val, test). Por tanto probaremos otros valores, los cuales son 60/20/20 y 50/25/25.

60/20/20:

```
trainbr
```

```
performance =
```

```
2.0799e-11
```

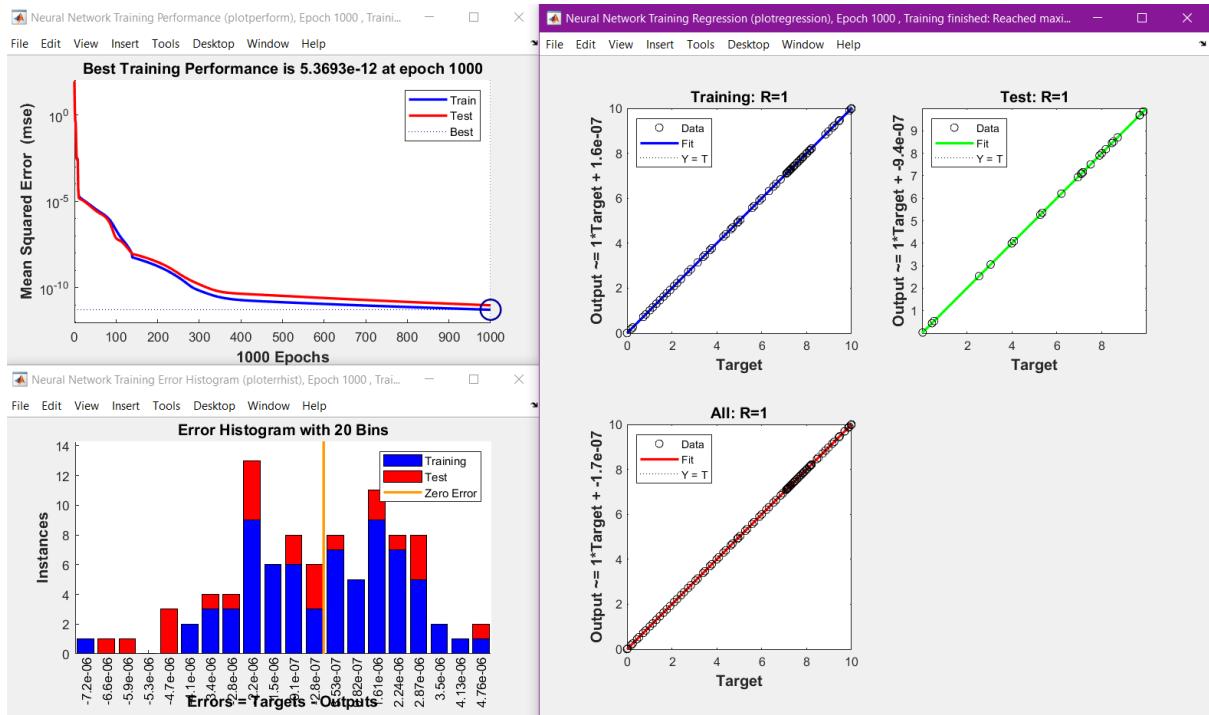


50/25/25:

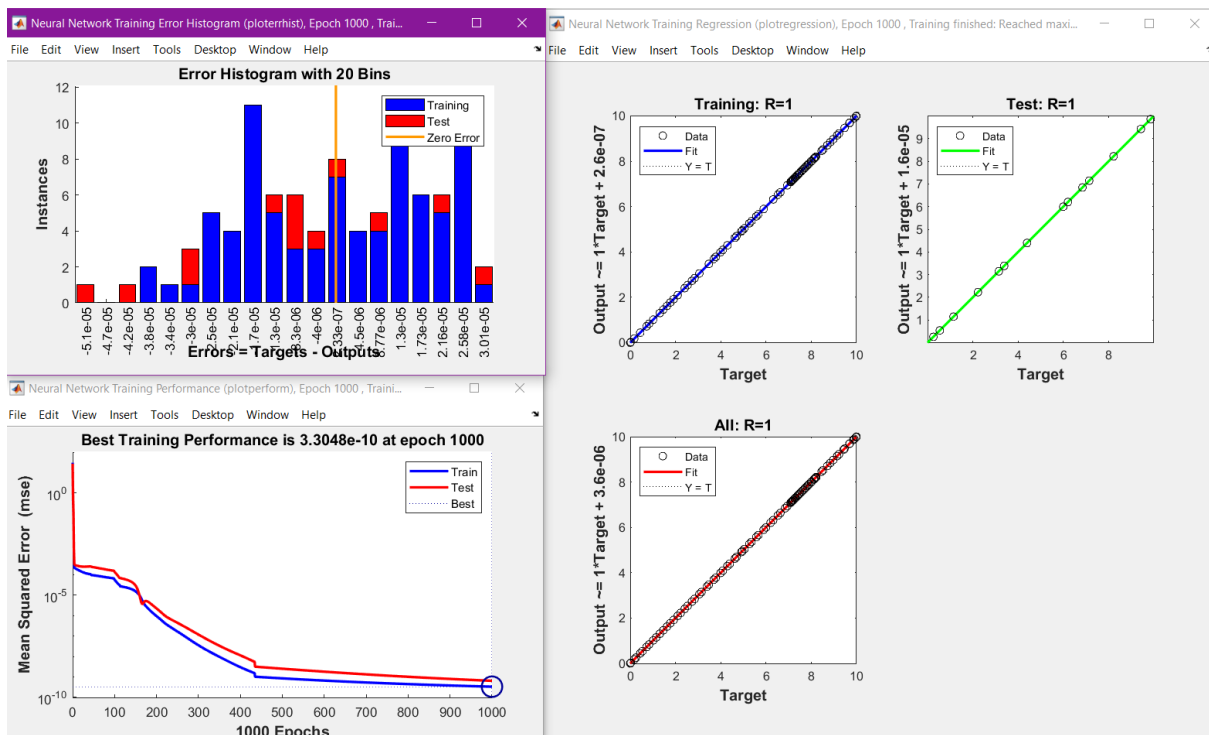
```
trainbr
```

```
performance =
```

```
6.4803e-12
```



70/15/15:



Se eligen para mostrar las gráficas de regresión, rendimiento e histograma de errores, ya que son los que más información útil arrojan. A partir de los gráficos y del rendimiento obtenido parece que cada vez que disminuimos la cantidad de valores de entrenamiento obtenemos un menor error y, en consecuencia, la red ajusta mejor. Es muy ligero el cambio de diferentes valores, casi insignificante. Lo podemos ver, por ejemplo, en las gráficas de regresión, que son prácticamente iguales, obteniendo un

muy buen resultado para cada valor, casi ideal. Asimismo, en las gráficas de histograma de errores y rendimiento, obtenemos prácticamente lo mismo, con la poca diferencia de que están un poco más ajustadas las líneas de rendimiento en el 50/25/25 que en el 70/15/15. Estos resultados nos dicen que, en este dataset, no importa casi la división de datos, hay otros factores más influyentes, al tratarse de una diferencia de rendimiento mínima. Otra conclusión que podemos sacar es que, al igual que antes, se ve que los datos son muy sencillos y no requieren de un excesivo entrenamiento, por lo que habría un “sobreajuste” mínimo con más datos de entrenamiento que hace que suba el error.

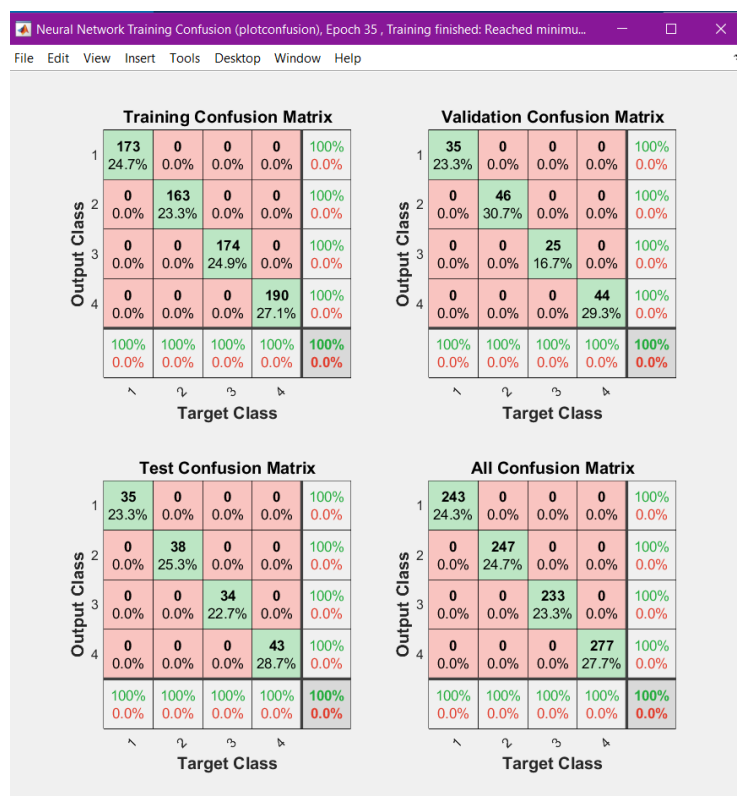
Ejercicio 4

Al igual que en el anterior ejercicio, tenemos un dataset simpleclass, que contiene un conjunto de datos sencillo para analizar, pero esta vez hay que clasificar. Seguramente, podamos sacar las mismas conclusiones que antes, comparando con un dataset que contiene datos más complejos: cancer_dataset. Vamos a verlo.

Simpleclass_dataset:

En la clasificación, se muestran las gráficas de Confusion y Receiver Operating Characteristic en lugar de regresión y fit.

Confusion muestra una matriz de confusión de los diferentes conjuntos de datos: entrenamiento, test, validación y todos. Se muestran los aciertos y fallos para cada clase con un eje en el que están las clases esperadas y en el otro las obtenidas.



En este dataset con la configuración dada por el problema que es de 10 neuronas de capa oculta, trainscg como método de entrenamiento predeterminado (cambia al ser patternnet en vez de fitnet, debido a que el algoritmo es mejor para clasificación que para regresión), la función de error es crossentropy, que pasa a ser la predeterminada al ser un problema de clasificación, y división de datos de 70/15/15.

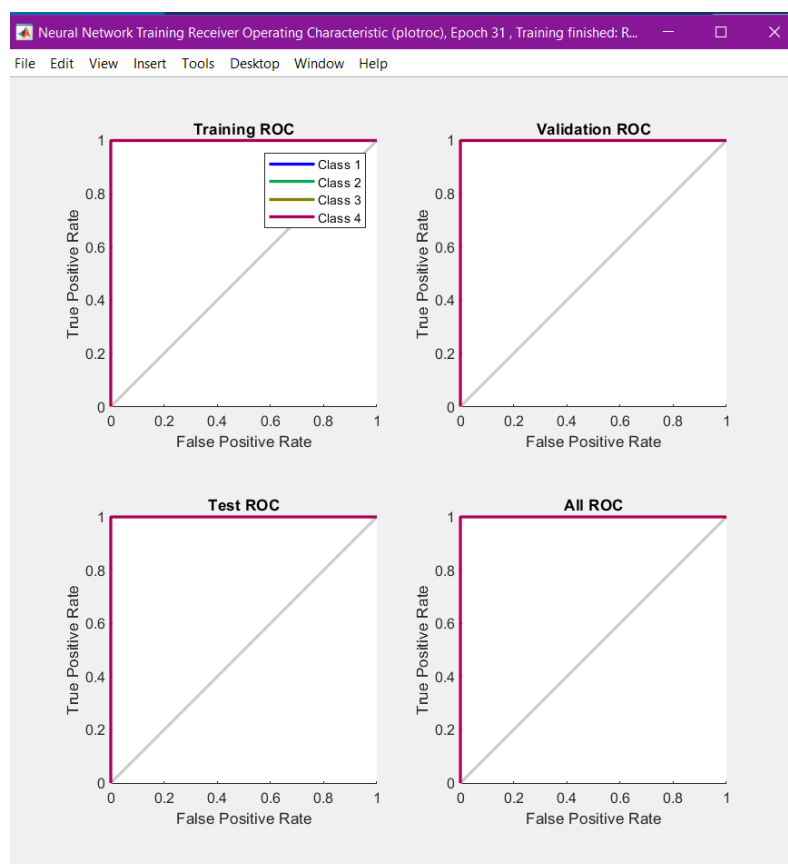
El rendimiento obtenido es:

```
performance =
```

```
8.7590e-07
```

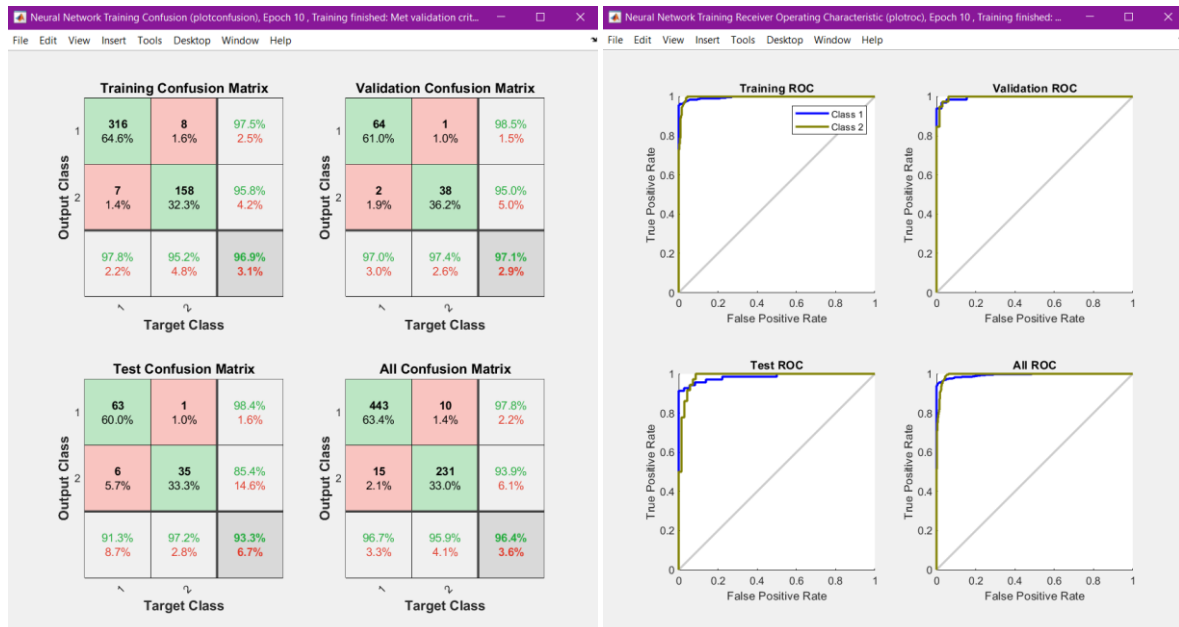
Pero más que el rendimiento en este dataset deberíamos mirar la matriz de confusión, ya que podemos obtener otro rendimiento si lo ejecutamos, pero serían los mismos resultados de clasificación. En la matriz de confusión hay un 100% de aciertos. Otra información que hemos detectado es que el dataset tiene 4 clases distintas.

Receiver Operating Characteristic nos muestra cuatro gráficos que representa la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos para diferentes.



Se puede ver que la línea se amolda perfectamente a la esquina de la izquierda al ser un resultado perfecto. Cuanto más se acerquen estas líneas de las clases a la esquina izquierda, mejor será el modelo.

Analizamos ahora el **dataset de cáncer**:



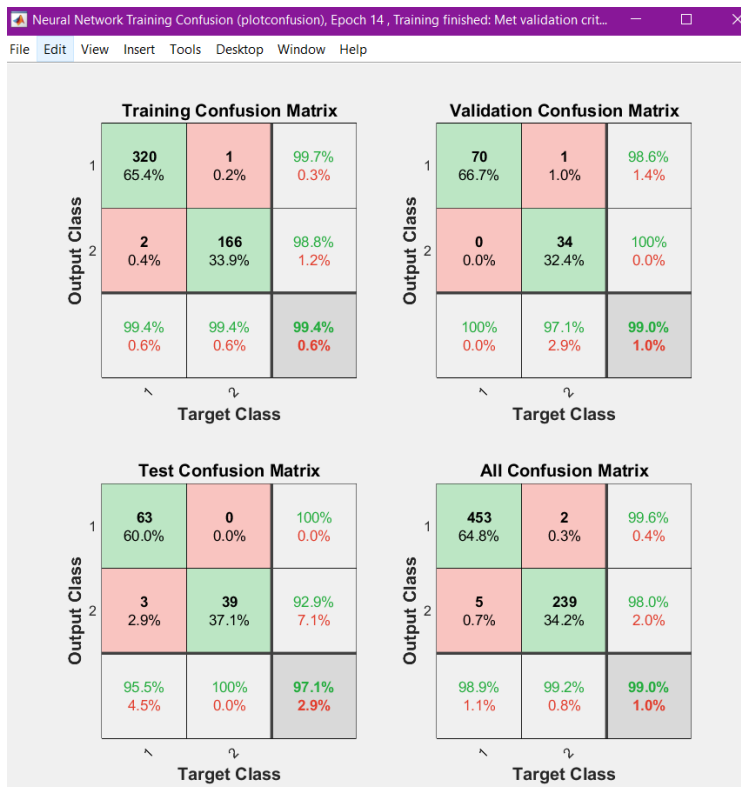
performance =

0.0497

Aquí tenemos otros resultados, un 96.9 % de acierto. Obtenemos un rendimiento mayor, debido a que, como se ve en la matriz de confusión, hay fallos de clasificación. En este dataset solo hay dos clases, sí tiene cáncer y no tiene cáncer. Además, hay más variables de entrada y los datos son más complejos.

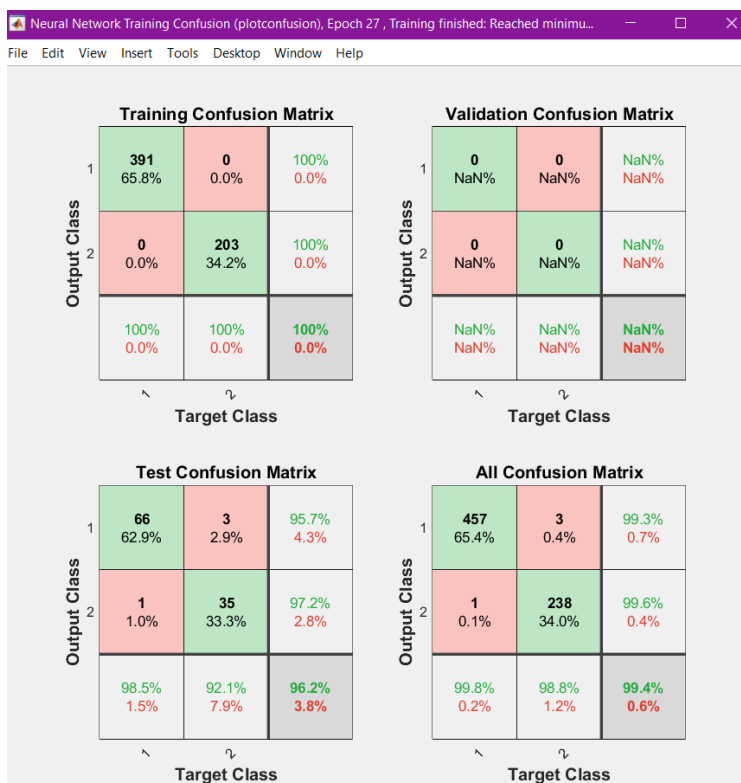
Vamos a probar otros métodos de entrenamiento para la dataset de cancer. Serán trainlm, trainbr y traingd, ya que trainscg es la de defecto ya analizada.

Trainlm:



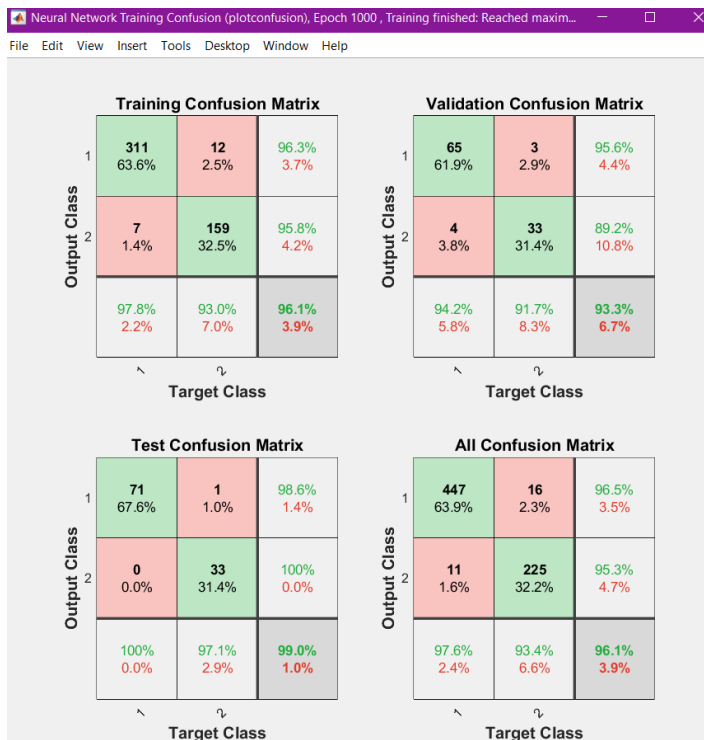
Trainlm clasifica correctamente el 99% con 33 épocas. Para trainlm y trainbr la función de error no es crossentropy, si no MSE.

Trainbr:

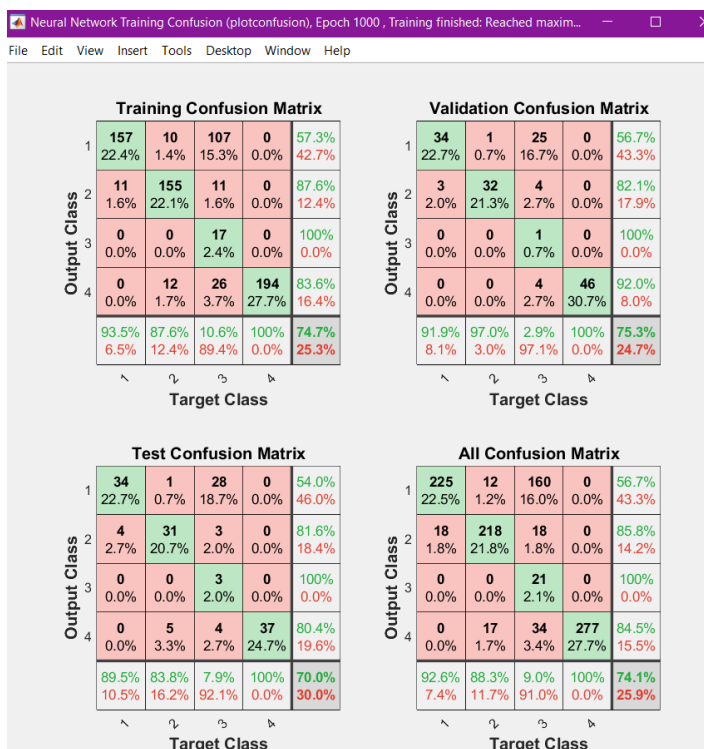


Trainbr clasifica bien un 99.4% con 27 épocas.

Training:



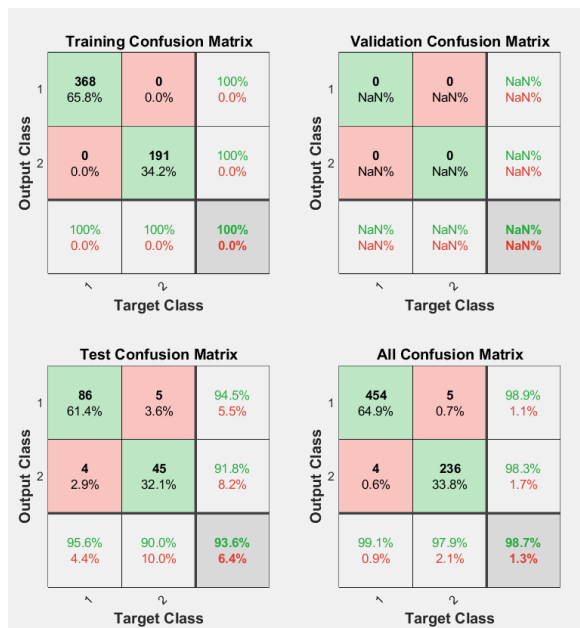
Trainngd saca correctamente un 96% con 1000 épocas.



Nos volvemos a quedar con la función trainbr.

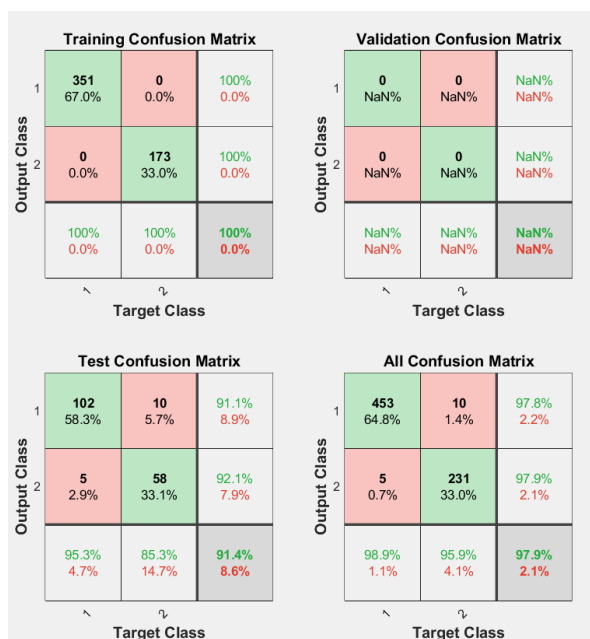
Probamos ahora la división de datos de entrenamiento. Como antes, se analizará 60/20/20 y 50/25/25.

60/20/20:



Se obtiene un peor resultado con un 98.7 % de aciertos. Lo único en lo que empeora es en test, ya que igual que con 70/15/15 hay 100% de aciertos.

50/25/25:

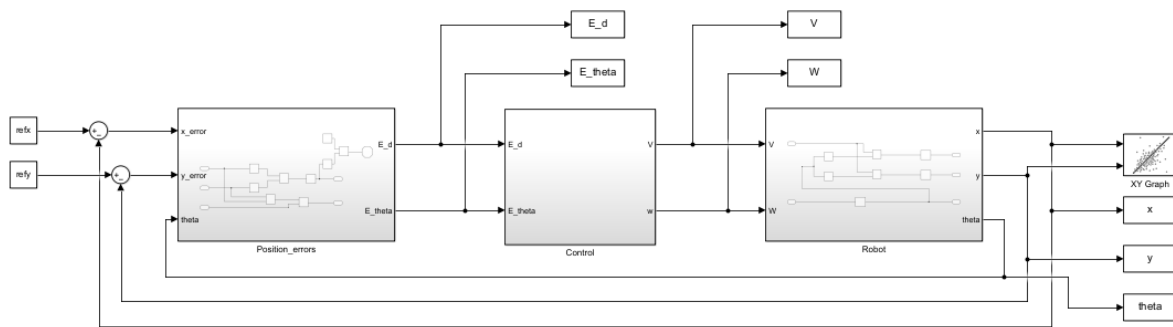


97.9% de aciertos. Cada vez peor.

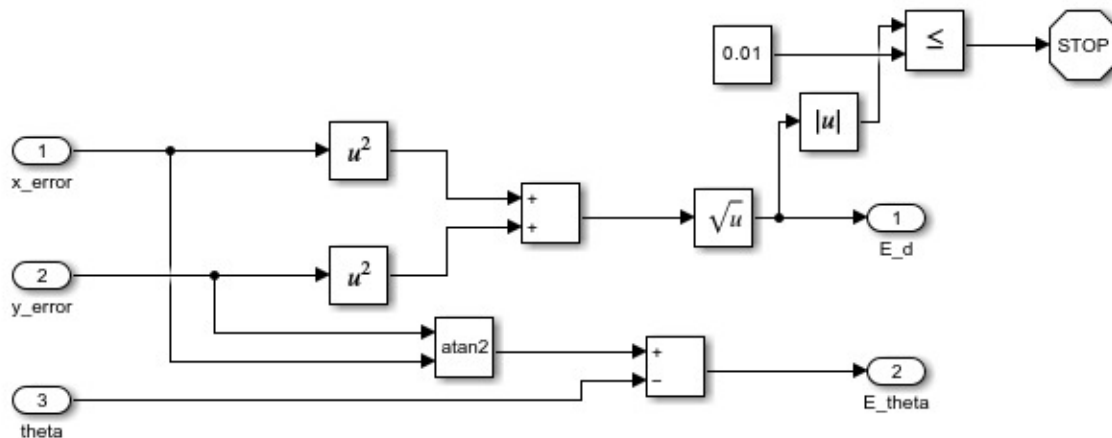
Esto indica que para este problema específico la mejor opción es 70/15/15 de división. Para datos más complejos es mejor tener más entrenamiento que un 50 o 60 por ciento. A su vez, no es correcto subir a un mayor número de entrenamiento, ya que podríamos caer en un problema de sobreajuste.

Segunda Parte

- a) Implemente el esquema de la Figura 1, incluyendo todos los bloques principales. Utilice como controlador el proporcionado en “controlblackbox.slx”. Configure los parámetros de la simulación (menú “Simulation/Model Simulation Parameters”) tal y como se muestra en la Figura 10. Guarde el esquema de Simulink con un nombre reconocible, por ejemplo “PositionControl.slx”.



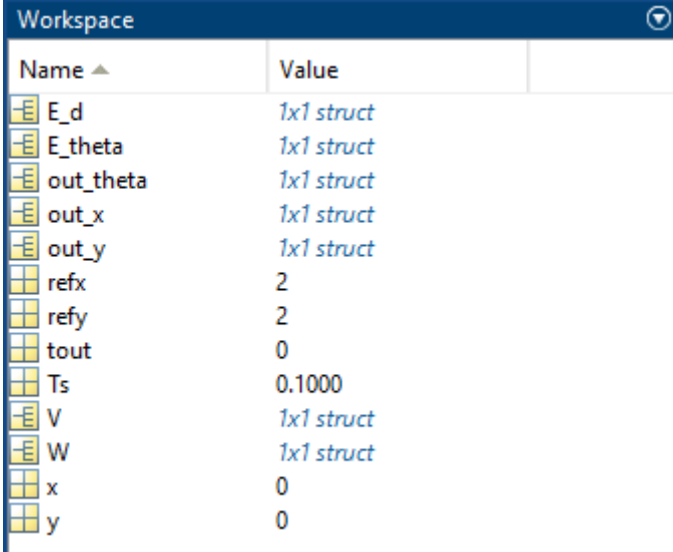
Position_errors:



- b) En el entorno de programación de Matlab, cree un script nuevo con el nombre “RunPositionControl.m”. Con el siguiente código se puede simular el diagrama PositionControl desde el entorno de comandos de Matlab, configurando el punto destino del robot mediante las variables refx y refy y el tiempo de muestreo Ts.

```
RunPositionControl.m
1 %Tiempo de muestreo
2 Ts=100e-3
3 % Referencia x-y de posicion
4 refx=2.0;
5 refy=2.0;
6 % Ejecutar Simulacion
7 sim('PositionControl.slx')
```

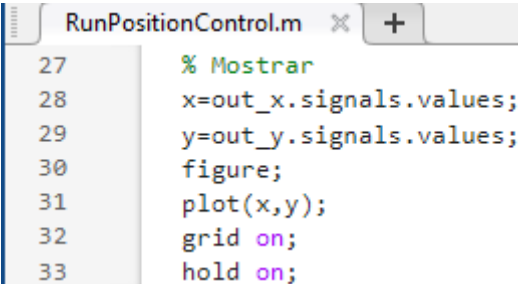
- c) Ejecute el script RunPositionControl y compruebe que se generan las variables que contienen las salidas y entradas del controlador (variables E_d E_theta V y W) y las salidas del robot durante la simulación (salida_x, salida_y, salida_theta).



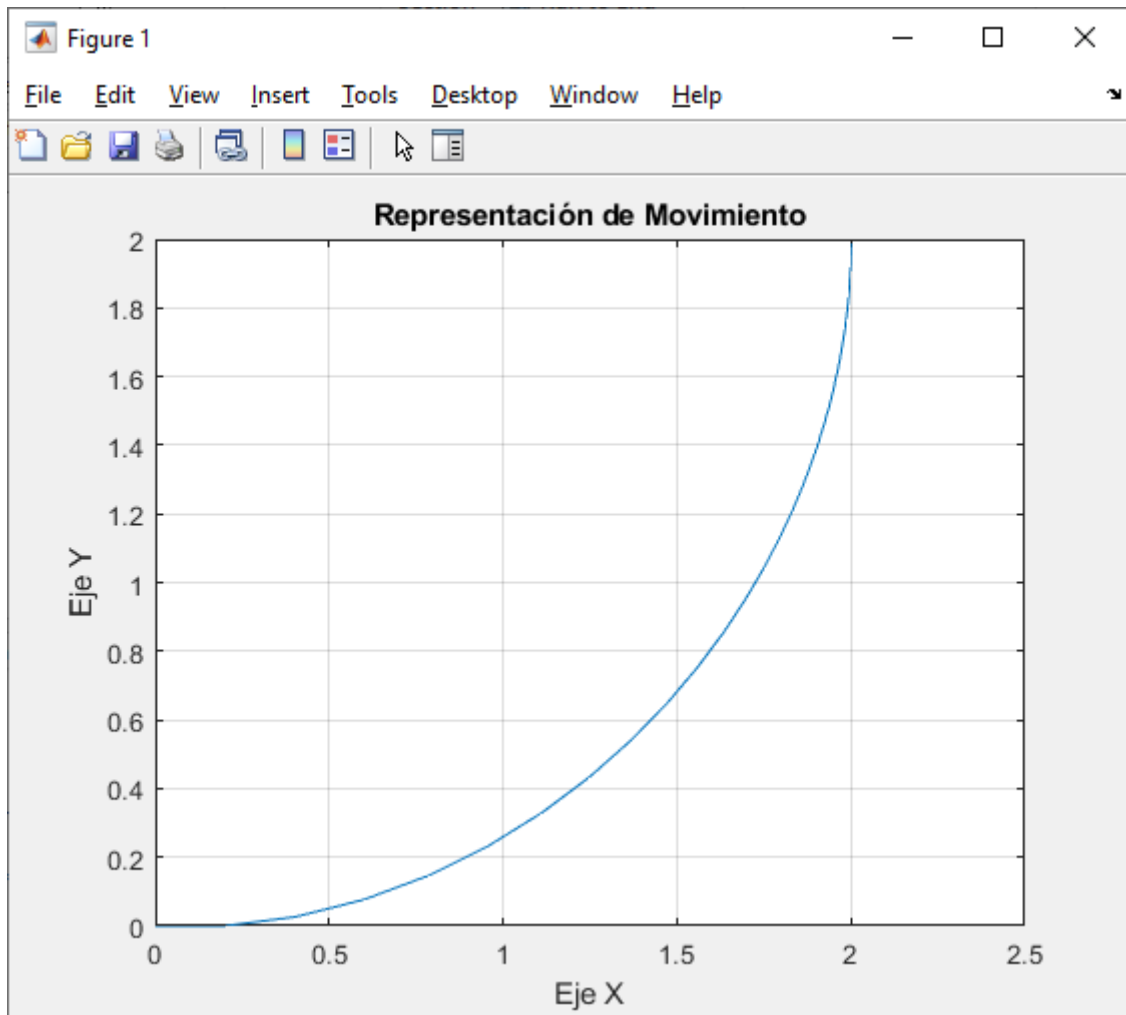
The screenshot shows the MATLAB Workspace window with a table of variables and their values. The variables are listed in the 'Name' column and their corresponding values in the 'Value' column.

Name	Value
E_d	1x1 struct
E_theta	1x1 struct
out_theta	1x1 struct
out_x	1x1 struct
out_y	1x1 struct
refx	2
refy	2
tout	0
Ts	0.1000
V	1x1 struct
W	1x1 struct
x	0
y	0

- d) Ejecute el siguiente código para mostrar la trayectoria del robot mediante el comando plot de Matlab.



```
RunPositionControl.m  x  +
27 % Mostrar
28 x=out_x.signals.values;
29 y=out_y.signals.values;
30 figure;
31 plot(x,y);
32 grid on;
33 hold on;
```

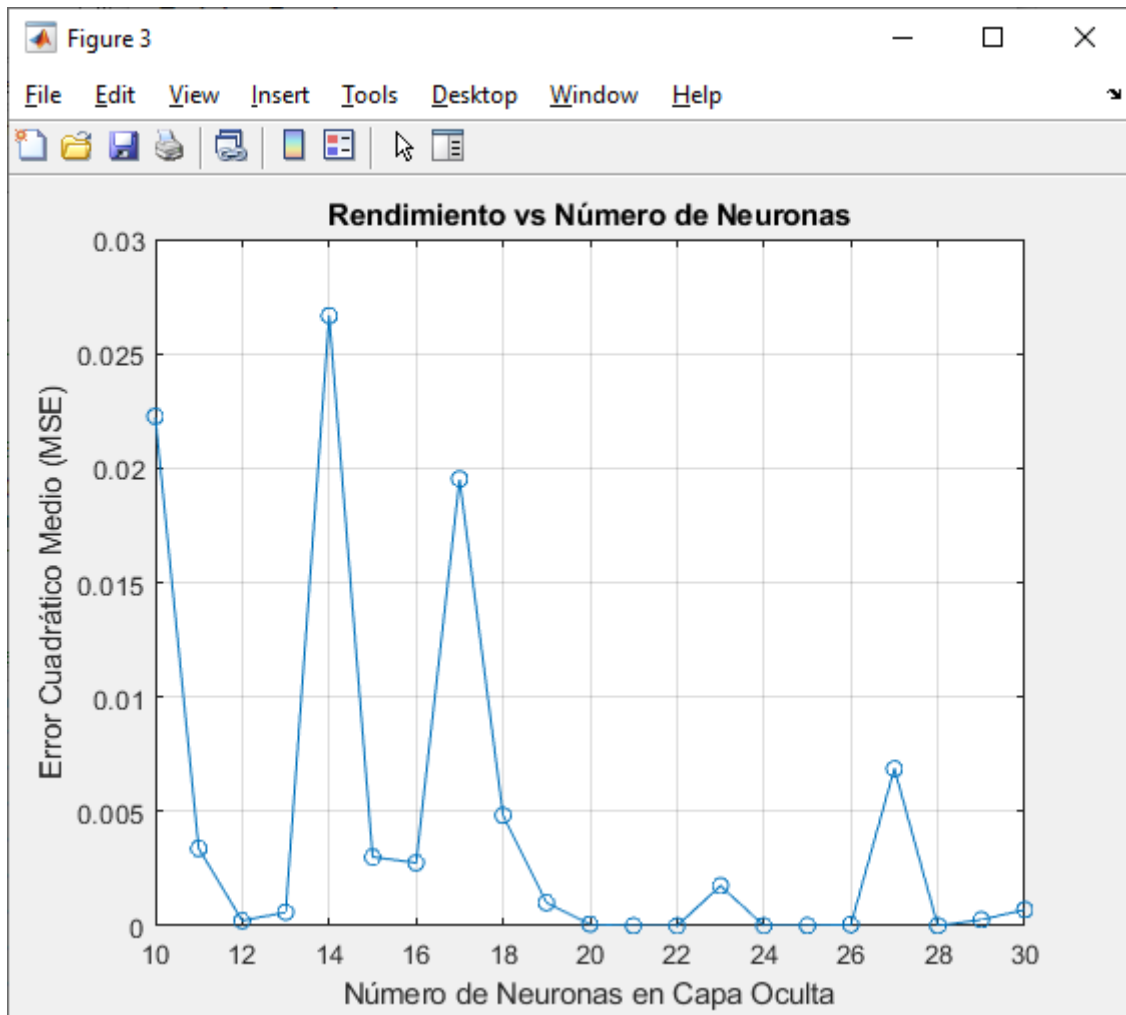



- e) Realice $N=30$ simulaciones del controlador proporcionado mediante un bucle donde se varían los valores de ref_x y ref_y de manera aleatoria dentro del entorno de 10×10 metros. En cada simulación se deberá guardar el valor a lo largo del tiempo de las entradas (E_d y E_{θ}) y salidas (V y W) del bloque controlador. Genere la matriz "inputs" de tamaño $2 \times N$, donde se acumulen los valores de E_d y E_{θ} . Del mismo modo genere la matriz "outputs", donde se acumulen los valores obtenidos de las variables V y W .

```
RunPositionControl.m  x  +
9      % Generar N posiciones aleatorias, simular y guardar en variables
10     N=30
11     E_d_vec=[];
12     E_theta_vec=[];
13     V_vec=[];
14     W_vec=[];
15     for i=1:N
16         refx=10*rand-5;
17         refy=10*rand-5;
18         sim('PositionControl.slx')
19         E_d_vec=[E_d_vec;E_d.signals.values];
20         E_theta_vec=[E_theta_vec;E_theta.signals.values];
21         V_vec=[V_vec; V.signals.values];
22         W_vec=[W_vec; W.signals.values];
23     end
24     inputs=[E_d_vec'; E_theta_vec'];
25     outputs=[V_vec'; W_vec'];
```

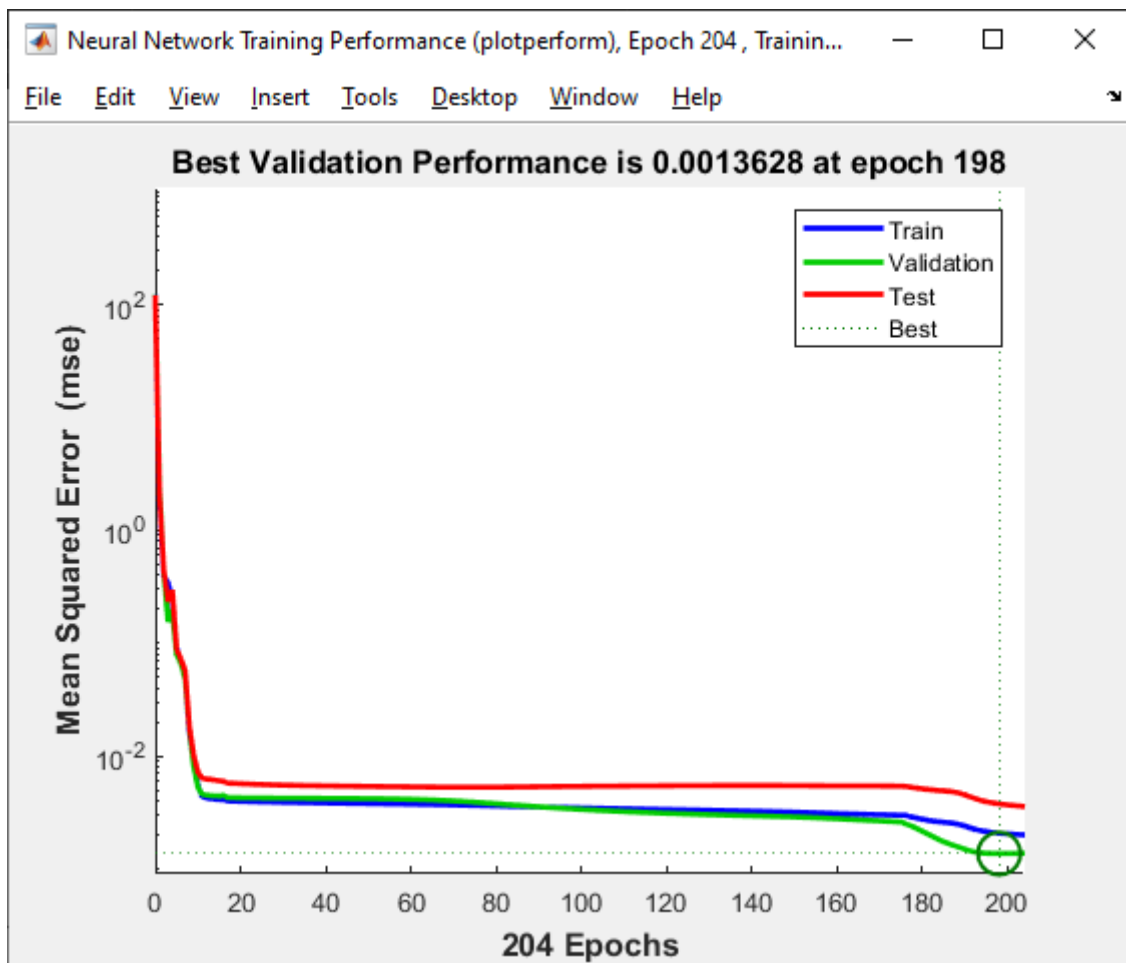
- f) Diseñe una red neuronal con una capa oculta de tal manera que dicha red se comporte como el controlador proporcionado. El número de neuronas de la capa oculta se deberá encontrar mediante experimentación. Justifique el valor elegido. El siguiente ejemplo muestra los comandos de Matlab para realizar dicho entrenamiento a partir de los vectores inputs, outputs.

La búsqueda del número óptimo de neuronas de la capa oculta ha llevado a la comparación de sus rendimientos en base a error cuadrático medio (MSE).



En la gráfica se puede ver que aumentar la complejidad de la red para lograr mejores resultados no es eficiente ya que se los errores son semejantes. Además, se puede llegar a sobreajustar la red. El número de neuronas elegido es de 12 en la capa oculta:

```
RunPositionControl.m x +
27 % Entrenar red neuronal con 12 neuronas en la capa oculta
28 net = feedforwardnet([12]);
29 net = configure(net,inputs,outputs);
30 net = train(net,inputs,outputs);
```



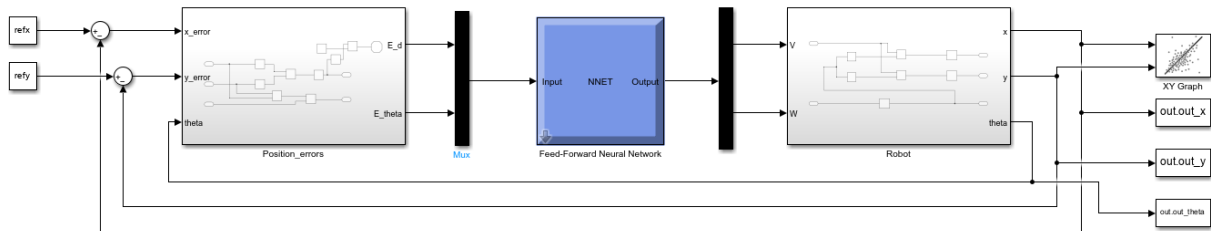
Al principio, el cambio en el rendimiento de la red es brusco ya que comienza con pesos aleatorios. A medida que van pasando las épocas y se van ajustando los pesos de la red, se obtienen mejores resultados en base al MSE. El entrenamiento de la red acaba cuando no se mejora el rendimiento o cuando se llega al número límite de épocas.

Con el entrenamiento, se mejoran los pesos de la red, con la validación se evita el sobreajuste de la red, y finalmente, con la prueba se determina el rendimiento de la red para datos nunca vistos.

- g) Genere, mediante la orden gensim de Matlab, un bloque con la red neuronal propuesta:

```
RunPositionControl.m x +
32 % Generar bloque de Simulink con el controlador neuronal
33 gensim(net,Ts)
```

- h) Cree un nuevo archivo de simulación “PositionControlNet.slx” con los mismos bloques utilizados en “PositionControl.slx” y donde se utilice la red neuronal en lugar del bloque controlador (Figura 12). Se utilizan bloques multiplexores y demultiplexores para adaptar las señales de entrada y salida como se muestra en la Figura.



- i) Compare el comportamiento de “PositionControlNet.slx” con “PositionControl.slx” para diferentes valores de ref_x y ref_y . Calcule el error entre las trayectorias realizadas por ambos controladores. Se recomienda realizar un script de Matlab para automatizar dicha comparación y mostrar los resultados.

```
RunPositionControl.m NetPerformance.m +
1 Ts = 100e-3;
2 refx_values = [-1, -3, 3];
3 refy_values = [-2, 1, 2];
4
5 figure;
6 subplot_counter = 1;
7
8 errors_x = [];
9 errors_y = [];
10
11 for i = 1:length(refx_values)
12     for j = 1:length(refy_values)
13         refx = refx_values(i);
14         refy = refy_values(j);
15
16         sim('PositionControlNet.slx');
17         sim('PositionControl.slx');
18
19         posX_net = out_x_net.signals.values;
20         posY_net = out_y_net.signals.values;
21
22         posX = out_x.signals.values;
23         posY = out_y.signals.values;
24
25         minLengthX = min(length(posX_net), length(posX));
26         minLengthY = min(length(posY_net), length(posY));
27
28         posX_net_trimmed = posX_net(1:minLengthX);
29         posY_net_trimmed = posY_net(1:minLengthY);
30         posX_trimmed = posX(1:minLengthX);
31         posY_trimmed = posY(1:minLengthY);
32
33         error_x = mean(abs(posX_net_trimmed - posX_trimmed));
34         error_y = mean(abs(posY_net_trimmed - posY_trimmed));
35
36         errors_x(end+1) = error_x;
37         errors_y(end+1) = error_y;
38
39         subplot(length(refx_values), length(refy_values), subplot_counter);
40         hold on;
41         plot(posX_net, posY_net, 'b-', 'DisplayName', 'Red Neuronal');
42         plot(posX, posY, 'r-', 'DisplayName', 'Controlador');
43
44         xlabel('Posición X');
45         ylabel('Posición Y');
46         title(sprintf('Refx: %.2f, Refy: %.2f', refx, refy));
47         legend('show', 'Location', 'West');
48         grid on;
49         hold off;
50
51         subplot_counter = subplot_counter + 1;
52     end
53 end
54
55 disp('Resumen de error medio absoluto');
56 for k = 1:length(errors_x)
57     fprintf('Para Refx: %.2f y Refy: %.2f -> Error en X: %.4f, Error en Y: %.4f\n', ...
58         refx_values(ceil(k / length(refy_values))), ...
59         refy_values(mod(k - 1, length(refy_values)) + 1), ...
60         errors_x(k), errors_y(k));
61 end
```

```
>> NetPerformance
```

Resumen de error medio absoluto

Para Refx: -1.00 y Refy: -2.00 -> Error en X: 0.0714, Error en Y: 0.0306

Para Refx: -1.00 y Refy: 1.00 -> Error en X: 0.0315, Error en Y: 0.0188

Para Refx: -1.00 y Refy: 2.00 -> Error en X: 0.0299, Error en Y: 0.0216

Para Refx: -3.00 y Refy: -2.00 -> Error en X: 0.2264, Error en Y: 0.0848

Para Refx: -3.00 y Refy: 1.00 -> Error en X: 0.1756, Error en Y: 0.0595

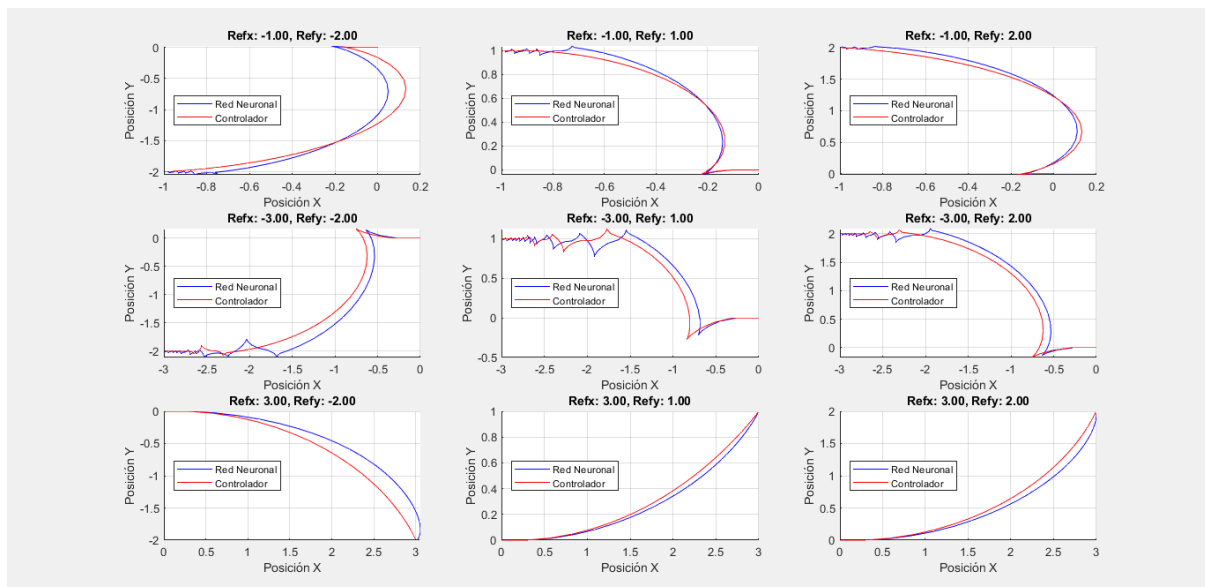
Para Refx: -3.00 y Refy: 2.00 -> Error en X: 0.1534, Error en Y: 0.0570

Para Refx: 3.00 y Refy: -2.00 -> Error en X: 0.0717, Error en Y: 0.0759

Para Refx: 3.00 y Refy: 1.00 -> Error en X: 0.0190, Error en Y: 0.0150

Para Refx: 3.00 y Refy: 2.00 -> Error en X: 0.0316, Error en Y: 0.0345

El error medio absoluto obtenido es del comienzo de las trayectorias ya que cada una tiene un tamaño distinto. La red neuronal no se asemeja del todo al controlador pero consigue sus mismos resultados:



Las gráficas muestran las diferencias entre la trayectoria proporcionada por la red neuronal y el controlador, combinando diferentes posiciones en el eje x y en el eje y. Se muestra que la funcionalidad de la red neuronal se asemeja a la del controlador, cumpliendo siempre la función de llegar al punto de referencia.

En conclusión, se puede crear un modelo que represente el actual sistema físico de una planta sin necesidad del modelo matemático de la misma como se ve en teoría. De esta manera, el sistema que se quiere replicar es una especie de caja negra pero se puede simular entrenando una red neuronal.