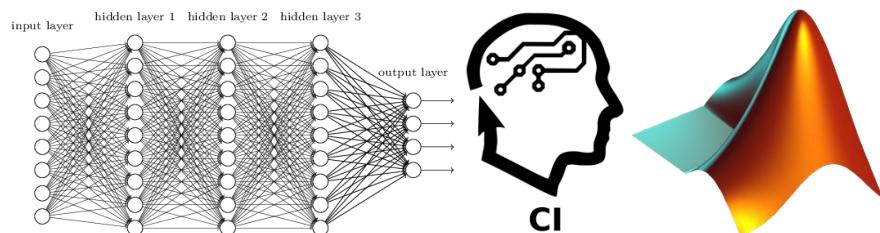


# Sistemas de control Inteligente

Ejercicios de Redes Neuronales

Juan José Córdoba Zamora  
Juan Casado Ballesteros



Universidad  
de Alcalá

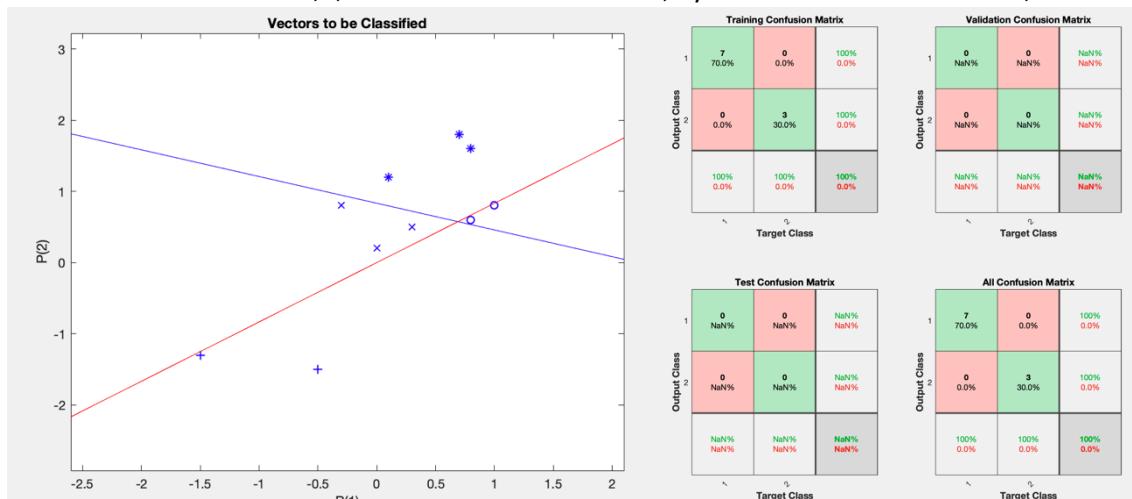
## ÍNDICE

EJ1P1 .....	3
EJ2P1 .....	6
EJ3P1 .....	20
EJ4P1 .....	26
EJ1P2 .....	34
EJ1P3 .....	37
EJ2P3 .....	40

## EJ1P1

Realizamos la clasificación de los datos proporcionados mediante una red de perceptores. Ya que tenemos 4 clases diferentes en las que los podemos clasificar la red nos generará 2 perceptrones que nos darán un total de  $2^n$  con  $n=2$  clases en las que poder clasificar los datos de entrada.

Para poder realizar la clasificación hemos tenido que binarizar las clases de salida, creando una matriz en este caso de  $m \times n$  con  $m$  el número de datos de entrada y  $n$  el log2 de las clases en las que queremos hacer la clasificación de modo que cuando un dato corresponda a la clase 0 creará una columna en la misma posición en la que estaba ese dato en la matriz de datos a clasificar con los valores de 0;0 ocurriendo esto mismo para el resto de las clases. La clase 1 creará una columna con los valores 1;0, la clase 2 con los valores 0;1 y la clase 3 con los valores 1;1.



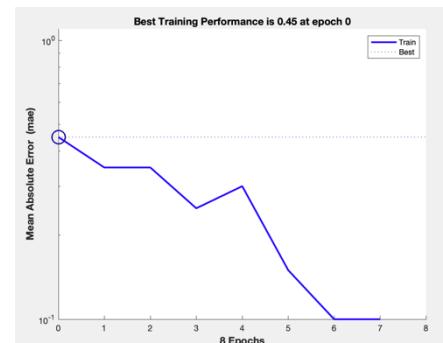
Tras entrenar la red con los datos proporcionados y entrenarla obtenemos los siguientes resultados en los que podemos ver como se ha logrado realizar la clasificación. Esto nos indica que nuestros datos de entrada eran linealmente separables respecto de las clases de salida proporcionadas. Cada dato perteneciente a cada clase está en una región del espacio distinta separada por las rectas que los perceptrones crean para separarlos. En las matrices de confusión vemos como todos los datos han sido clasificados correctamente sin falsos positivos ni negativos.

Debido a lo reducidos que eran nuestros datos de entrada no hemos separado estos en datos de validación ni test de modo que todos eran datos de entrenamiento. La red se ha aprendido por tanto estos datos sin mayor dificultad debido al hecho de que ha podido hacerlo pues eran linealmente separables.

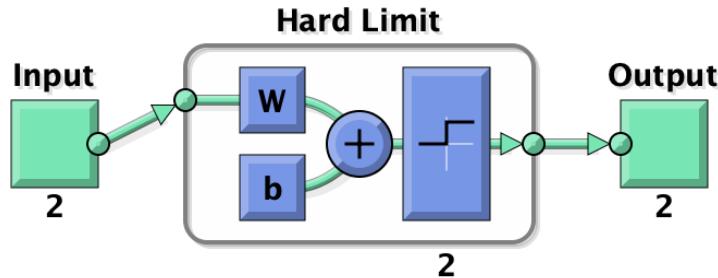
Progress			
Epoch:	0	8 iterations	1000
Time:		0:00:00	
Performance:	0.450	0.00	0.00

Algo que hemos podido observar es que siempre que la red logra clasificar los datos nunca llega al límite de épocas máximo, por el contrario, cuando la clasificación no es posible si se llega a él.

Vemos también como el error cae en picado sobre valores muy pequeños en la gráfica de error, esto se debe a que se han logrado clasificar todos los datos correctamente y que el conjunto de datos a clasificar era muy reducido.



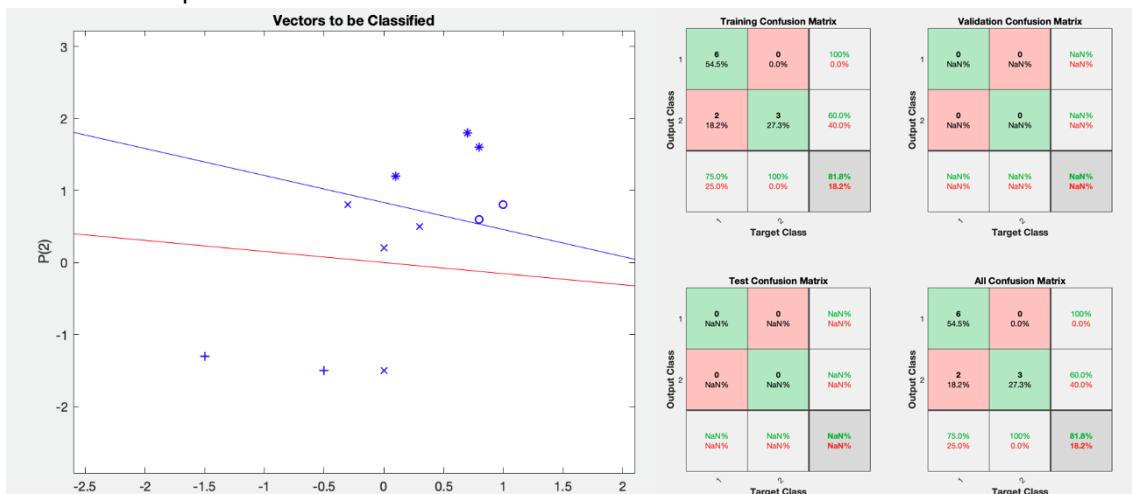
Finalmente tal y como hemos mencionado al principio vemos que la red tiene dos neuronas de salida lo cual se debe al número de clases en las que debe clasificar los datos. Las salidas de los perceptrones son binarias y necesitamos dos salidas para poder alcanzar las 4 clases en las que los datos deben ser clasificados



Incorporamos ahora un nuevo dato a nuestro conjunto de datos a clasificar. Este nuevo dato se encuentra en la región de los datos de la clase 1 pero le decimos a la red que es de la clase 3 de modo que convertimos nuestros datos en un conjunto que no es linealmente separable.

Por las razones indicadas anteriormente, seguimos teniendo 4 clases en las que clasificar nuestros datos seguiremos teniendo dos perceptrones en nuestra red y 4 salidas posibles representadas en dos neuronas de salida binarias.

Podemos observar ahora que la clasificación no llega a realizarse pues los datos no son linealmente separables.



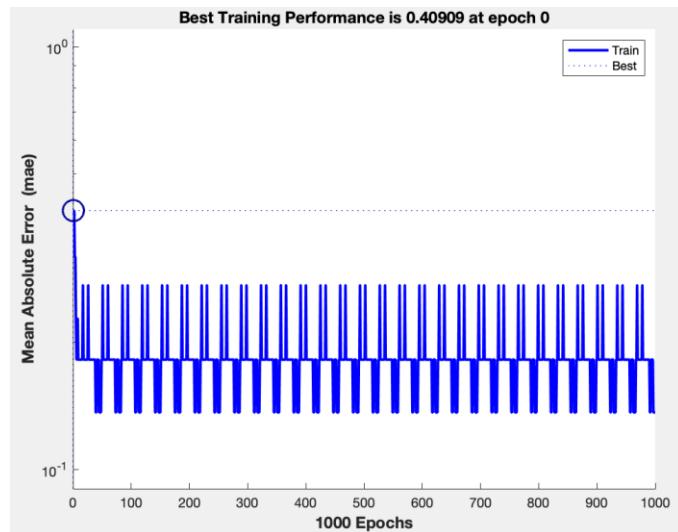
Las rectas por las que se divide el plano no clasifican correctamente todos los datos y por tanto en la matriz de confusión vemos como aparecen dos valores que no han sido correctamente clasificados.

En este punto cabe preguntarse por qué si solo uno de los datos resulta ser no linealmente separable (el último introducido) aparecen más de uno que no han sido clasificados correctamente. Esto se debe a que la linealidad es una propiedad del conjunto de datos y no de cada uno por separado de modo que es el conjunto al completo el que no puede ser clasificado por esta red en concreto. Podemos confirmar esto realizando el entrenamiento varias veces de modo que obtendremos distintas clasificaciones posibles todas ellas con más de un error.

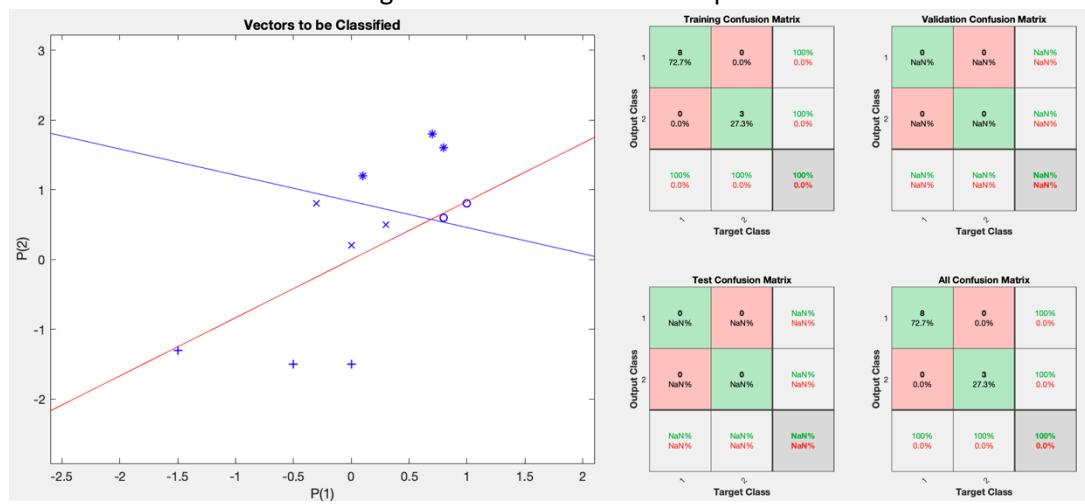
Progress			
Epoch:	0	1000 iterations	1000
Time:		0:00:06	
Performance:	0.409	0.136	0.00

Como ya mencionamos anteriormente que se llegue al límite de iteraciones posibles para entrenar la red es un indicador de que la clasificación no se ha realizado correctamente.

Adicionalmente podemos ver como el error en esta clasificación no ha caído si no que ha empezado a oscilar lo cual nos indica que los datos no han podido separarse linealmente pero que la red ha seguido intentando hacerlo.



Por último, podemos confirmar que si el nuevo elemento no hubiera sido elegido con tanta malicia y hubiera sido asignado a la clase 1 la clasificación si se hubiera podido realizar pues en este caso los datos hubieran seguido siendo linealmente separables.



## EJ2P1

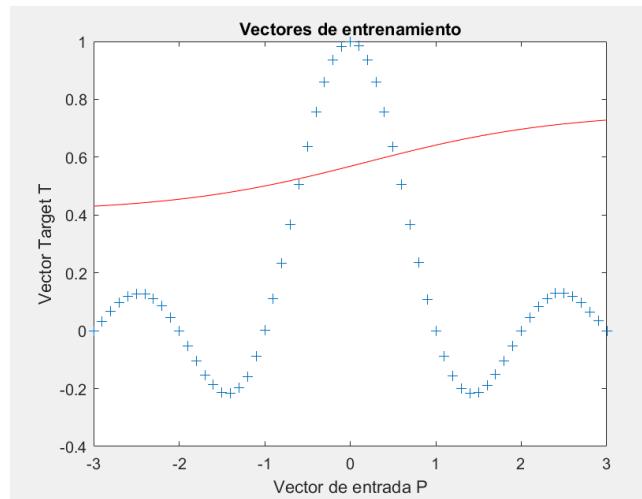
Para este ejercicio se van a utilizar los métodos de entrenamiento trainrp, trainbr, trainlm, trainbfg y traincgb. Para cada método de entrenamiento se cambiarán el número de neuronas de la capa oculta por un valor muy pequeño, otro normal y otro muy grande para poder comparar los resultados.

### Trainrp:

Este método de entrenamiento se llama Resilient backpropagation, actualiza los valores de peso y sesgo de acuerdo con el algoritmo de retropropagación resistente (Rprop).

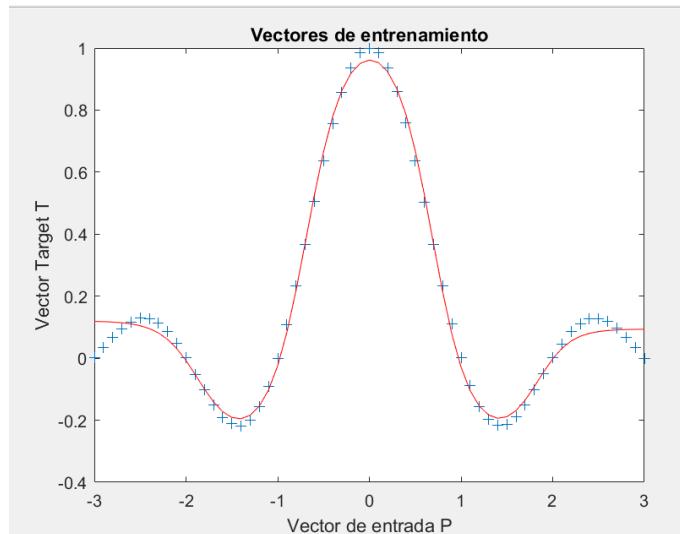
- **Numero de neuronas de la capa oculta = 1**

Si ponemos el número de neuronas de la capa muy bajo, como por ejemplo 1, se observa que la red no consigue aprender y por lo tanto el resultado obtenido es que la función resultado no se ajusta nada a la función esperada.



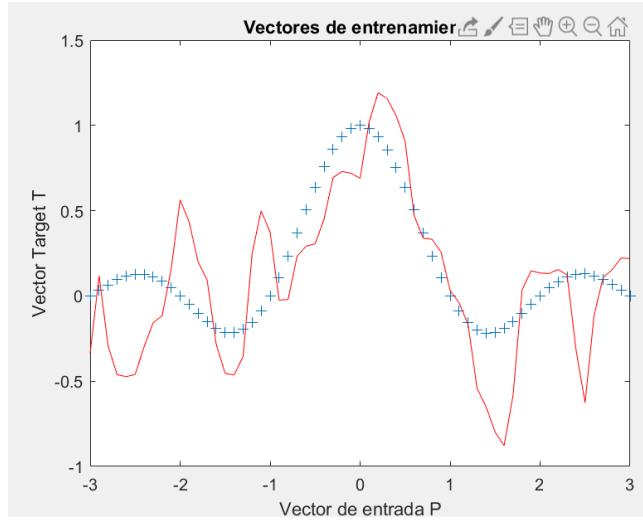
- **Numero de neuronas de la capa oculta = 4**

Si ponemos un número de capas ocultas intermedio, de 4 a 8, la red consigue aprender y dar unos resultados bastante bueno, pero no es capaz de ajustarse perfectamente a la función que queremos aproximar.

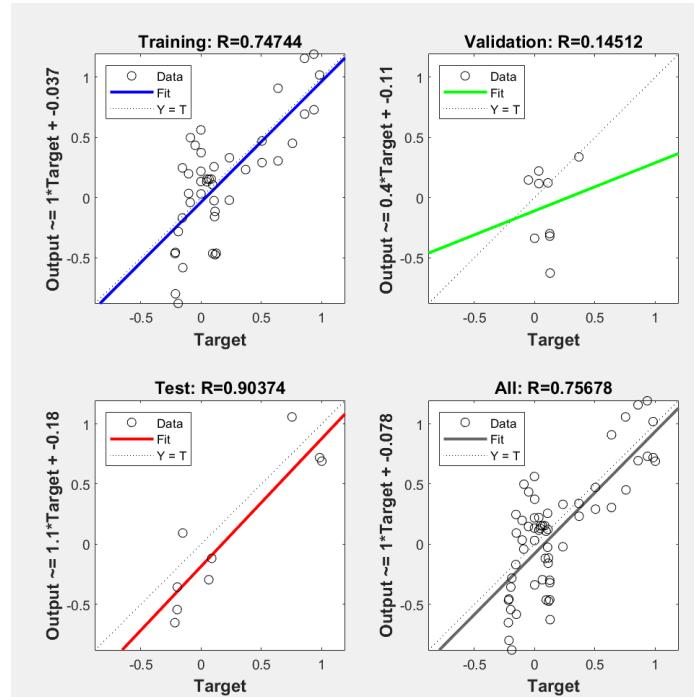


- **Numero de neuronas de la capa oculta = 30**

Si ponemos un número de neuronas alto, más de 20, se observa que la red aprende demasiado todos los datos de entrenamiento y luego con los datos de test y validación no es capaz de ajustarse a la función y por lo tanto ofrece resultados con muchos picos y nada ajustados a los puntos de la función para aproximar.



Esto se puede comprobar en la gráfica de regression, que como se puede observar los puntos no siguen la línea definida con un ángulo de 45 grados, lo que significa que es una mala generalización.

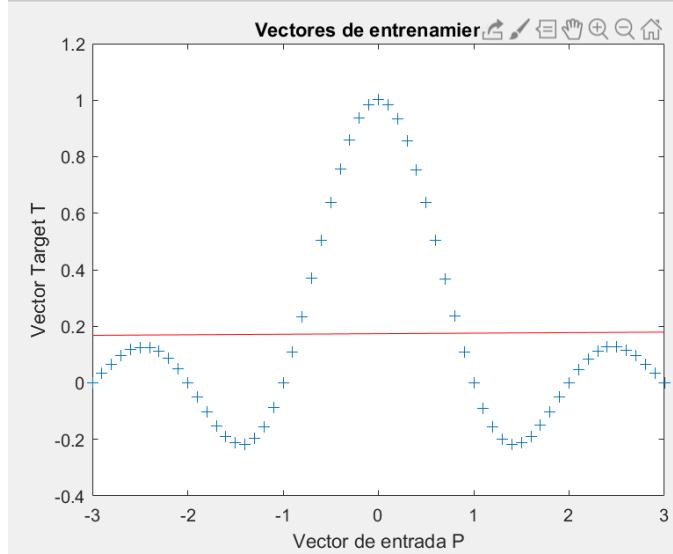


#### **Trainbr:**

Este método de entrenamiento se llama Bayesian regularization backpropagation, actualiza los valores de peso y sesgo según la optimización de Levenberg-Marquardt.

- **Numero de neuronas de la capa oculta = 1**

Como se observa en el gráfico que devuelve la red neuronal, el resultado es una línea recta y por lo tanto no se ajusta a la función, esto significa que la red neuronal no consigue aprender, ni siquiera para los datos de entrenamiento.

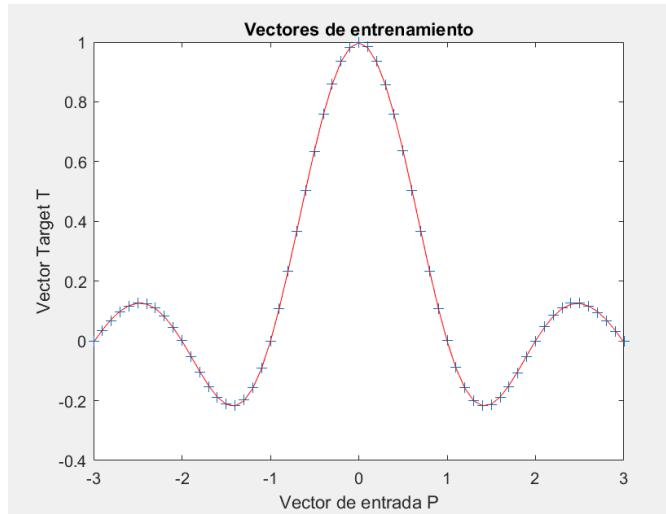


Si mostramos el error se puede observar que el error baja únicamente al principio y luego se mantiene estable, por lo que como hemos indicado anteriormente la red no consigue aprender.



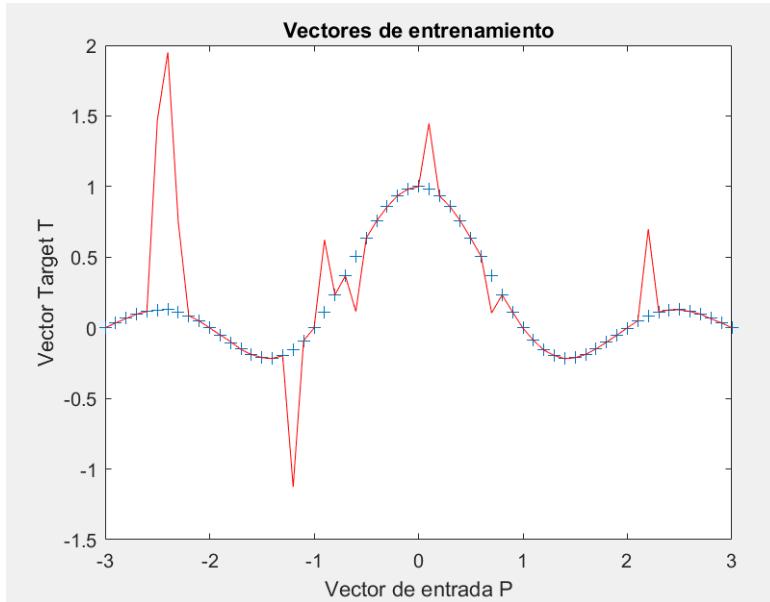
- **Numero de neuronas de la capa oculta = 4**

Con un número de neuronas intermedio se puede observar que la red consigue aprender y es capaz de ajustarse a la función perfectamente. También hemos realizado esta prueba con otro número de neuronas, 8, 20 y 50 y para este rango de números también era capaz de ajustarse perfectamente a la función resultado.



- **Numero de neuronas de la capa oculta = 100**

Si variamos el número de neuronas y ponemos un valor muy alto el resultado de la red neuronal es que no se ajusta a la función, sino que aparecen picos por toda la función resultado. Esto es así porque la red aprende demasiado los datos de entrenamiento y luego para los datos de validación y test no es capaz de predecir buenos resultados.

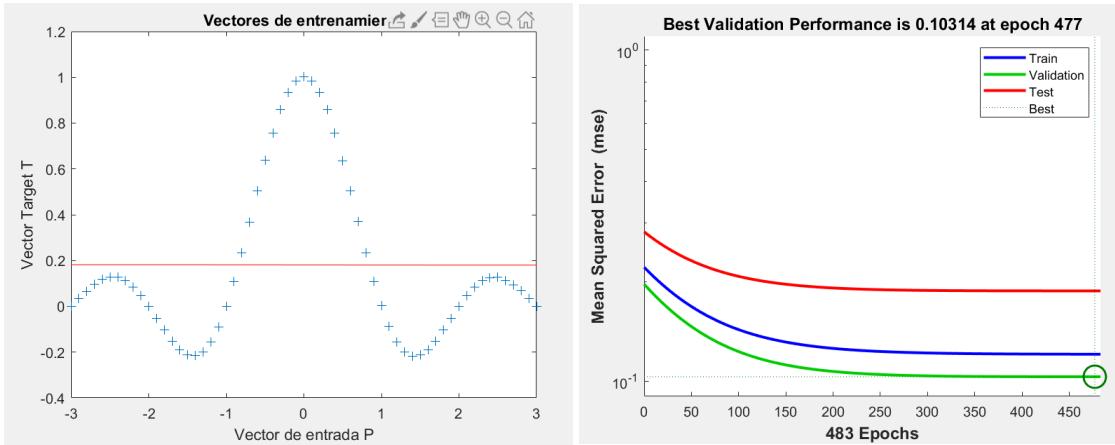


Para este método de entrenamiento ha sido necesario poner un número muy grande de neuronas, ya que si no poníamos ese número el entrenamiento lo hacía de forma correcta, por ejemplo, al poner 30 conseguía seguir la trayectoria de forma muy buena.

**Traingd:** Este método se llama Gradient descent backpropagation, actualiza los valores de peso y sesgo según el descenso del gradiente.

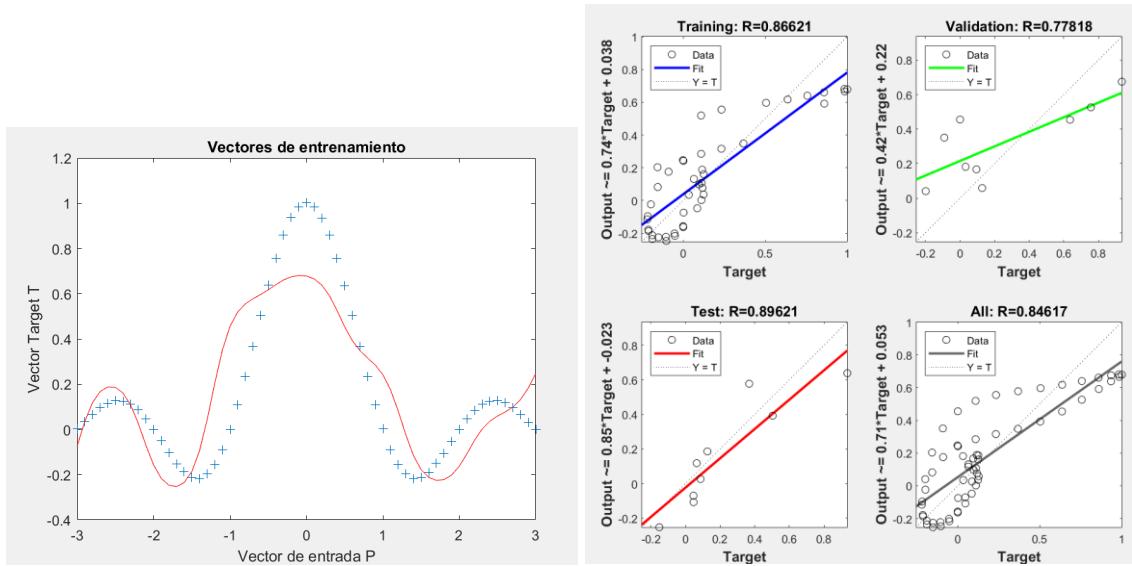
- **Numero de neuronas de la capa oculta = 1**

Si indicamos un número tan bajo de neuronas, la red neuronal no será capaz de entrenar de forma correcta, por lo que, si intentamos predecir con esta red, el resultado es una línea recta. No se pudo entrenar de forma correcta. Como podemos ver el error empieza a decrecer muy poco al principio, pero en seguida comenzó a mantenerse, por lo que hemos comentado anteriormente la red no ha sido capaz de aprender.



- Numero de neuronas de la capa oculta = 8**

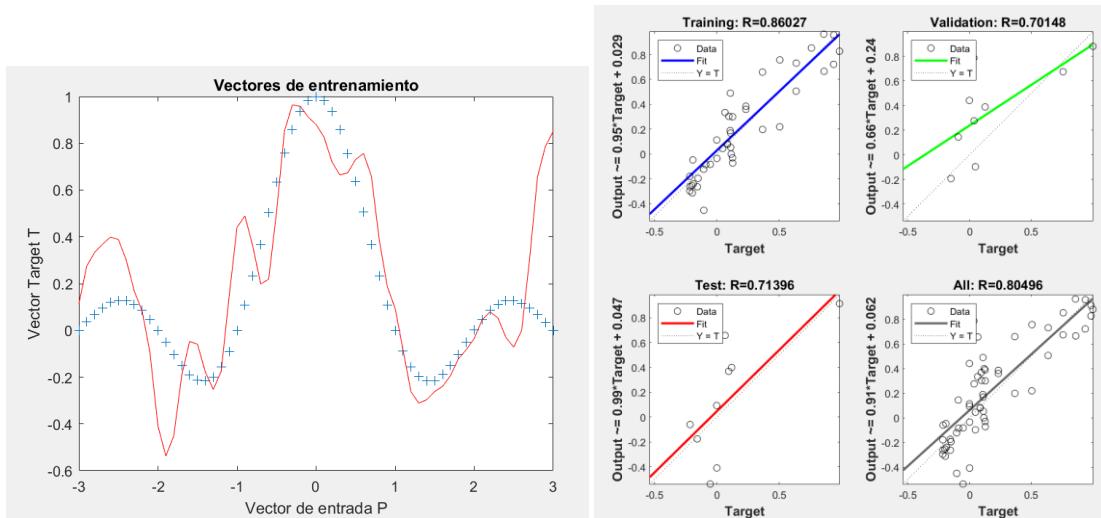
Si ponemos un número de neuronas medio en la capa de oculta, por ejemplo 8, aunque hemos probado de 5 a 10 y los resultados eran muy parecidos.



Como se observa en los gráficos anteriores no ha sido capaz de predecir correctamente la función, es decir, la red no ha sido capaz de aprender de forma óptima. En la gráfica de regresión se aprecia como los datos no han sido correctamente clasificados para ninguno de los subconjuntos de datos, incluso para los de entrenamiento.

- Numero de neuronas de la capa oculta = 20**

Si indicamos un número de neuronas muy alto, el resultado que obtenemos es que la red se sobreentrena y por lo tanto los resultados que ofrecen no son correctos, la red no es capaz de generalizar los conocimientos de los datos de entrenamiento a los datos de test.

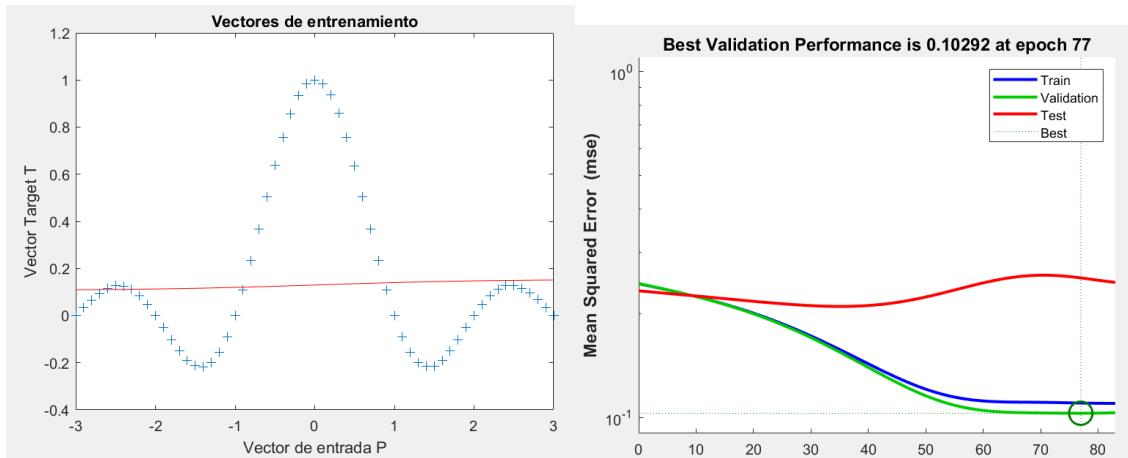


En estas gráficas se puede observar como los resultados obtenidos no encajan en la función que queríamos aproximar, se pueden ver picos en algunos puntos y esto es debido al sobreentrenamiento de la red, no ha sido capaz de predecir ni generalizar para los datos de test. En la gráfica de regresión se puede ver como los datos no han seguido una línea recta de 45 grados, esto es por lo que hemos comentado anteriormente, no han sido predichos de forma correcta.

**Traingdx:** Este método se llama Gradient descent with momentum and adaptive learning rate backpropagation, actualiza los valores de peso y sesgo de acuerdo con el momento de descenso del gradiente y una tasa de aprendizaje adaptativo.

- **Numero de neuronas de la capa oculta = 1**

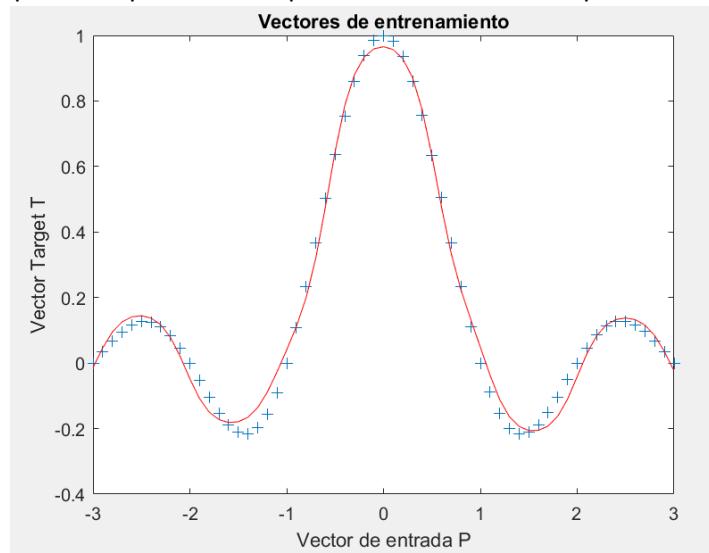
Si indicamos un número de neuronas muy bajo, se observa que la red no es capaz de aprender un por lo tanto el resultado es una línea recta.



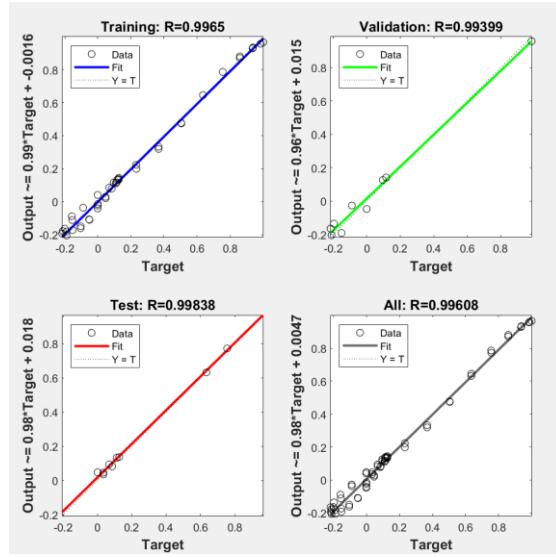
En las dos gráficas anteriores se puede ver cómo el resultado es una línea y el error ha decrecido muy poco al principio, pero enseguida se ha mantenido y no se ha reducido más el error.

- **Numero de neuronas de la capa oculta = 8**

Si indicamos un número de neuronas medio, se observa como la red se ajusta más a los puntos que se tiene que aproximar pero no es capaz de hacerlo de forma perfecta.



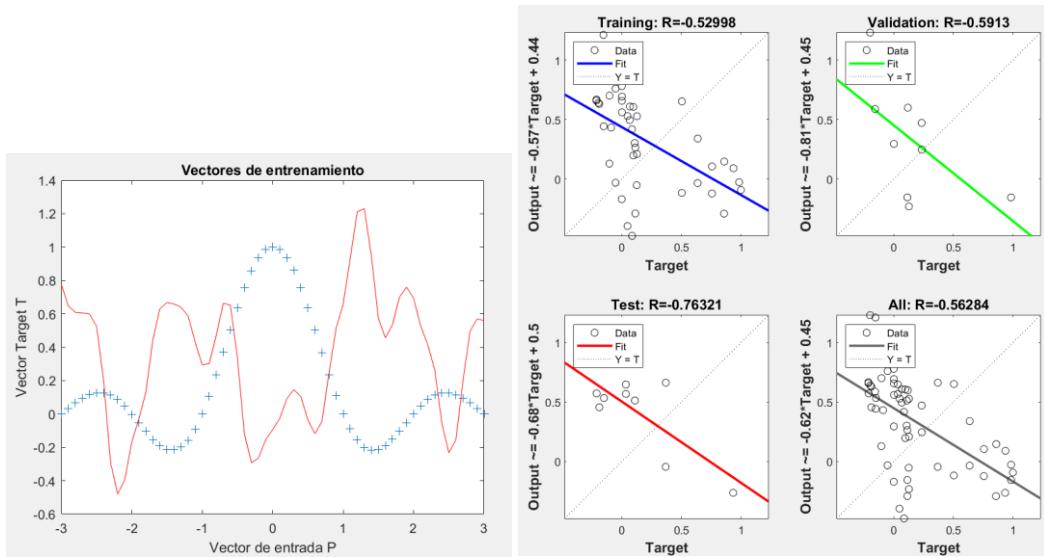
Aunque como se puede observar el error es mínimo, la trayectoria la consigue seguir de forma correcta.



Si observamos la gráfica de regresión vemos como los datos se encuentran en línea recta formando 45 grados, esto se debe a que la red ha sido capaz de generalizar los datos de entrenamiento y dar un muy buen resultado.

- **Numero de neuronas de la capa oculta = 20**

En cambio, se ponemos un número alto de neuronas en la capa oculta, se observa como se producen picos en los puntos al intentar predecirlos, esto es así porque la red no es capaz de generalizar los datos de entrenamiento, se los aprende de forma perfecta y al intentar predecir con otros datos no es capaz.

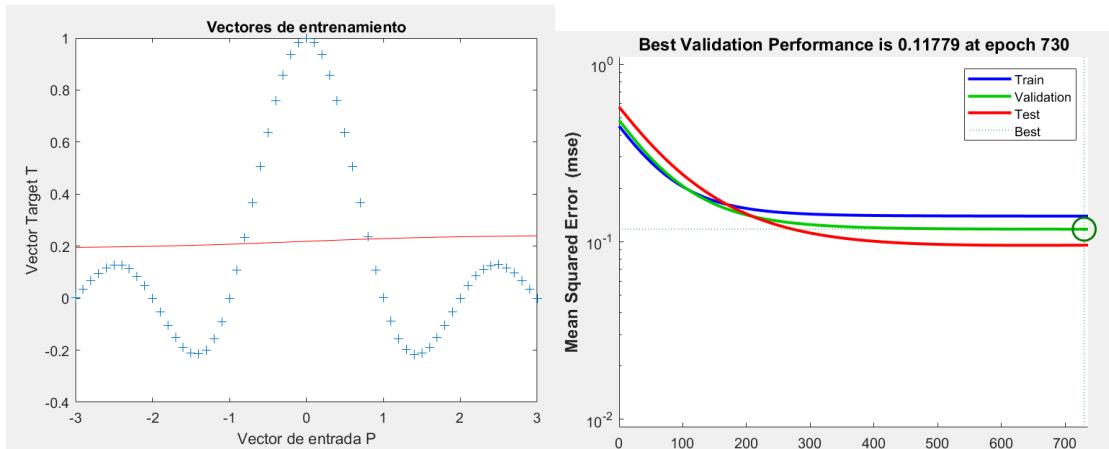


Como se puede ver en la gráfica de regresión la recta que sale no tiene 45 grados de inclinación y por lo tanto los datos se encuentran bastante lejos de lo que debería ser correcto.

**Traingdm:** Este método se llama Gradient descent with momentum backpropagation,

- **Numero de neuronas de la capa oculta = 1**

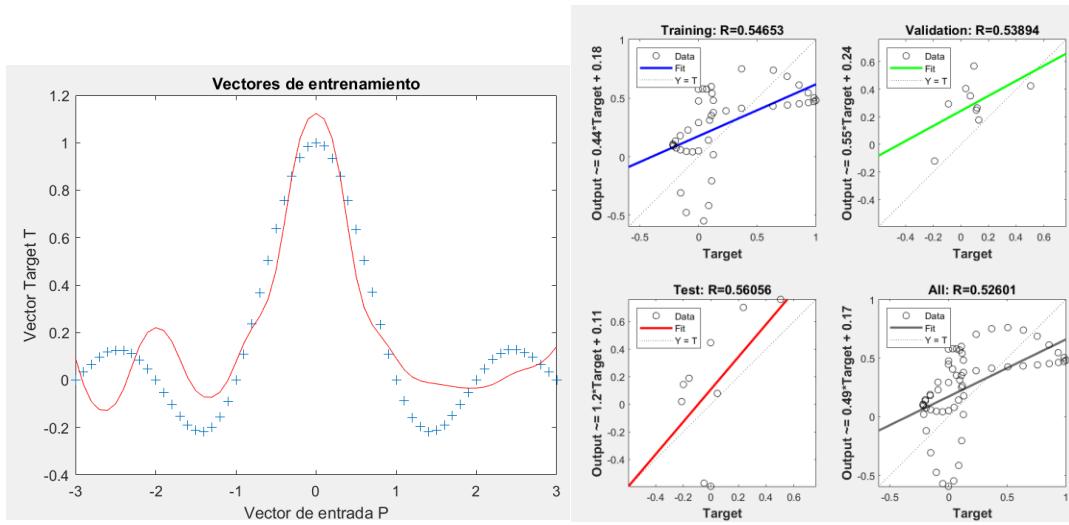
Si utilizamos un número muy bajo de neuronas como hemos comprobado con los demás métodos la red no es capaz de aprender y por lo tanto el resultado es una línea recta.



El error empieza a disminuir al principio, pero enseguida se mantiene.

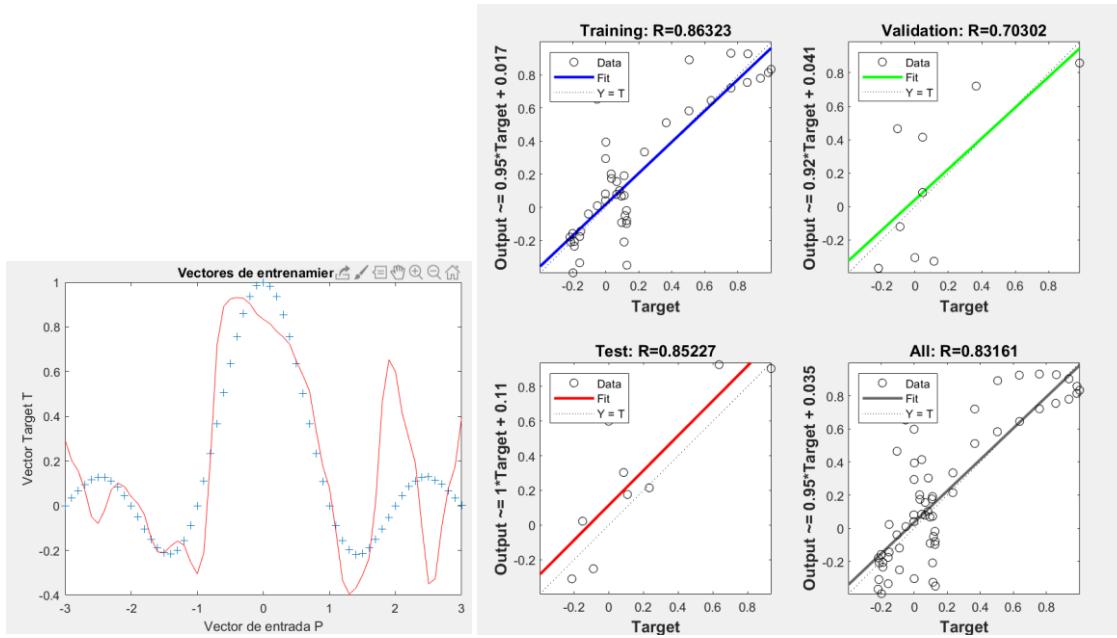
- **Numero de neuronas de la capa oculta = 10**

Con un número medio de neuronas en la capa oculta, es cuando la red obtiene los mejores resultados, aunque estos resultados están muy lejos de ser correctos, solo es capaz de seguir la trayectoria que sigue la función, aunque no se aproxima a ningún punto. En la gráfica de regresión se puede ver como los puntos que salen están muy separados y no forman una recta de 45 grados.



- **Numero de neuronas de la capa oculta = 20**

Si indicamos un número muy alto de neuronas en la capa oculta, la red no es capaz de generalizar los resultados, por lo que el resultado son picos al intentar aproximar la función. Esto se debe a que la red se aprende perfectamente el conjunto de datos de entrenamiento.



Y en la gráfica de regresión se observa como los puntos no forman una recta de 45 grados, estos puntos se encuentran bastante lejos de la recta.

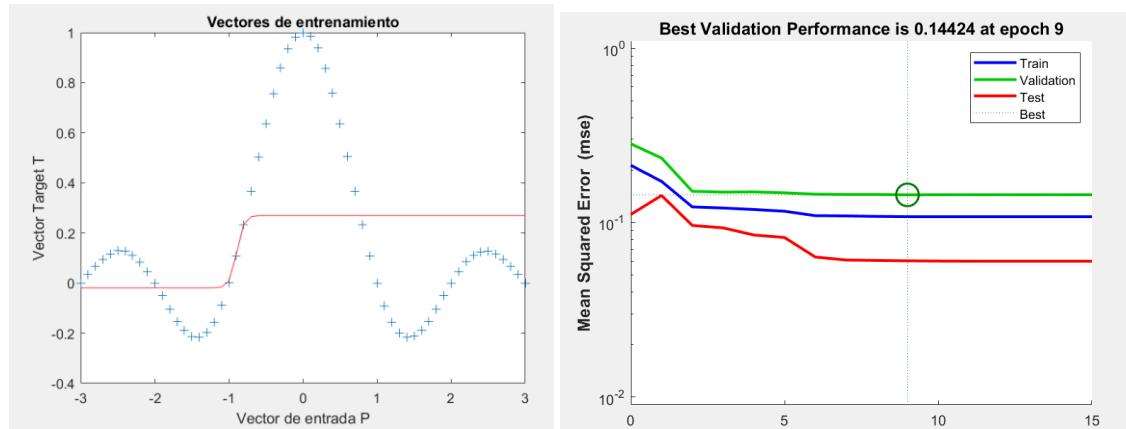
#### Trainlm:

Este método se llama Levenberg-Marquardt backpropagation, actualiza los valores de peso y sesgo de acuerdo con la optimización de Levenberg-Marquardt.

- **Numero de neuronas de la capa oculta = 1**

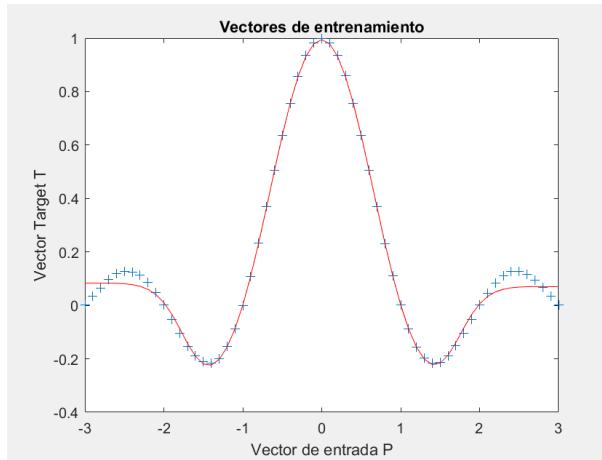
Para un número de neuronas muy bajo se observa que la red consigue aprender un poco al principio (el error de entrenamiento decae), pero llega un punto en el que se mantiene. En cambio, para los datos de test y validación el error decae muy poco al principio, pero en seguida se mantiene constante. Esto da como resultado que la función se ajuste en tres puntos, pero en

general la red neuronal no ha aprendido y el resultado de ésta son dos líneas rectas pasando por los 3 puntos indicados.

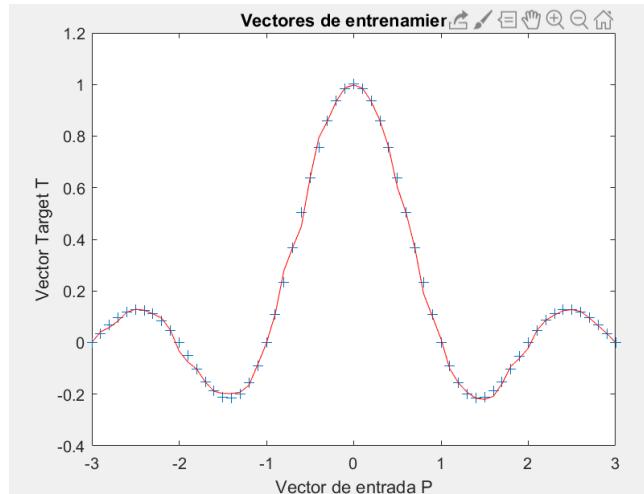


- Numero de neuronas de la capa oculta = 4**

Para un número de neuronas intermedio, se observa que el resultado se ajusta casi perfectamente a la función que queremos aproximar, existen algunos puntos que no es capaz de aproximar, pero el resultado en general es bastante bueno y el error de los puntos que no se ajusta no es grande.

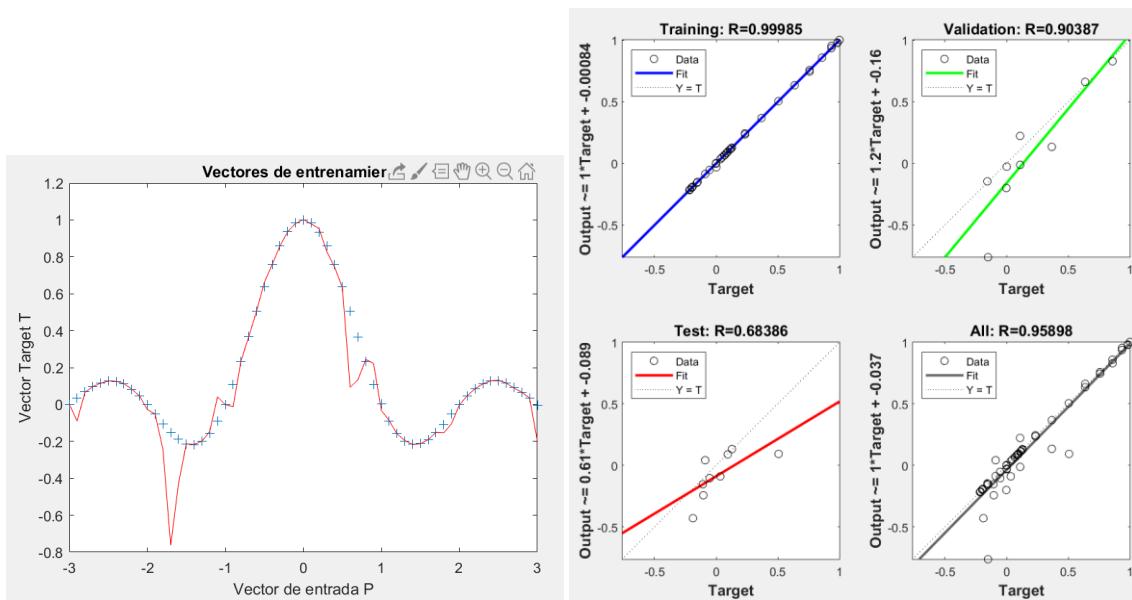


Si ponemos el número de neuronas a 20, se observa que el resultado es mucho más preciso y se ajusta mejor a los puntos, pero el resultado de la red no es tan suave y da pequeños picos.



- **Numero de neuronas de la capa oculta = 40**

Para un número de neuronas de la capa oculta mayor de 30 se observa como la red sobreaprende y al intentar ajustarse a la función de aproximación, en la red se muestran picos que indica que la red se ha aprendido demasiado bien los datos de entrenamiento, pero para los datos de validación y test no ha sido capaz de ofrecer buenos resultados.

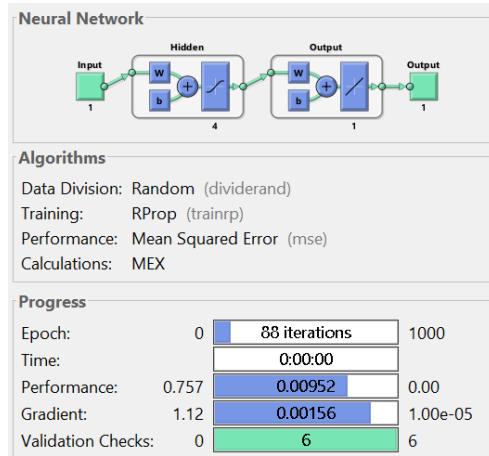


En la gráfica de regresión se observa como los datos de entrenamiento los domina y predice perfectamente, pero al intentarlo con los datos de validación y test se ve como no es así, los puntos que predice se encuentran bastante alejados del resultado que tenía que salir.

Si comparamos el número de neuronas de la capa oculta observamos que, si indicamos un número muy bajo 1, la red no es capaz de aprender y si indicamos un número muy alto, generalmente más de 20 aunque depende del método de entrenamiento, la red aprende demasiado y luego no se ajusta perfectamente a la función produciendo picos. Por lo que un número medio de neuronas es lo correcto para que la función sea capaz de aprender y ajustarse sin problemas a los puntos de la función, generalmente este número es entre 4 y 15 aunque dependiendo del método de entrenamiento variará el resultado.

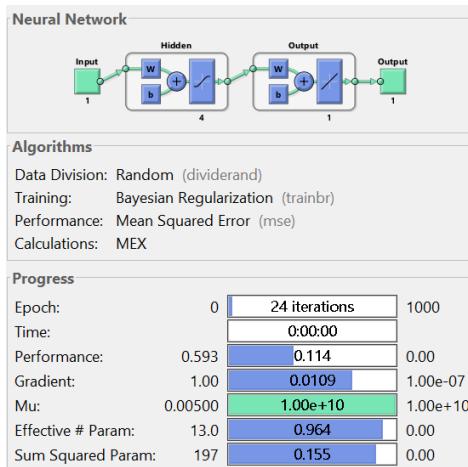
Para cada método vamos a comparar el rendimiento de cada método, para ello vamos a utilizar para todos los métodos el número de neuronas que nos ha dado mejores resultados y comparar resultados como puede ser el número de iteraciones, el performance y el tiempo medio.

### Trainrp:



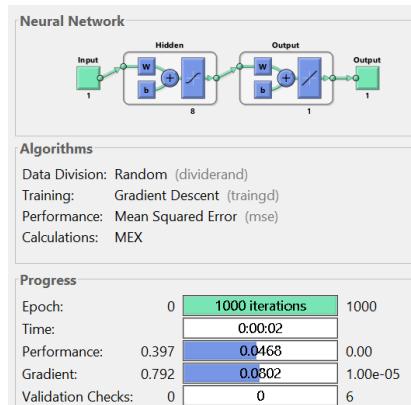
El performance de este método es bastante bueno, el número de iteraciones es muy bajo y al entrenar la red no ha tardado nada, ha sido instantáneo.

### Trainbr:



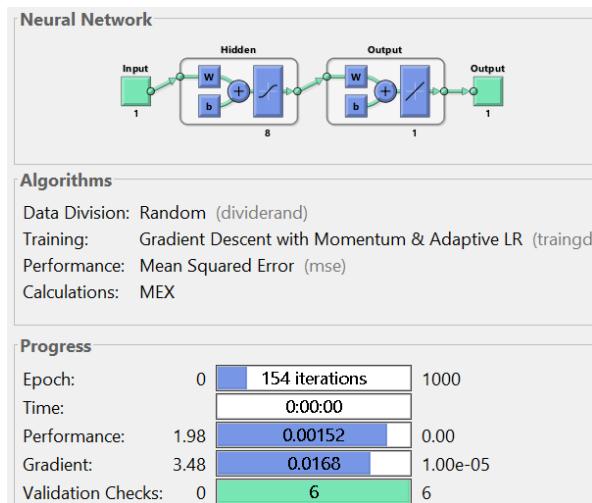
Para este método de entrenamiento se puede observar cómo el tiempo de entrenamiento y el número de iteraciones es muy bueno, pero el performance es alto.

### Traingd:



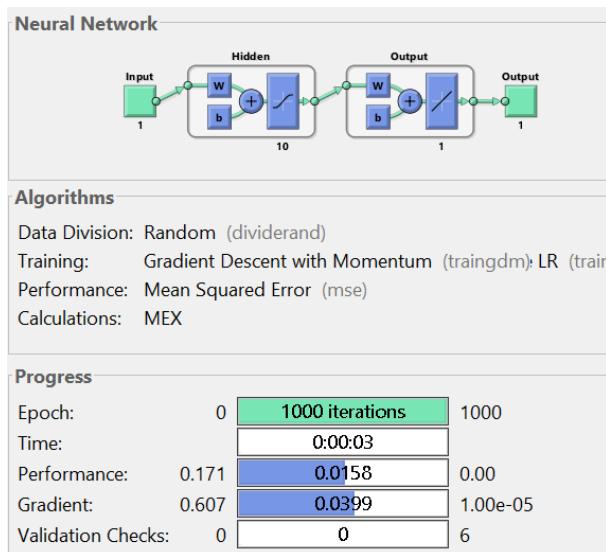
Para este método de entrenamiento se puede observar cómo agota todas las iteraciones, además se observa cómo tarda un poco más que los anteriores, el performance es relativamente bajo.

#### Traigdx:



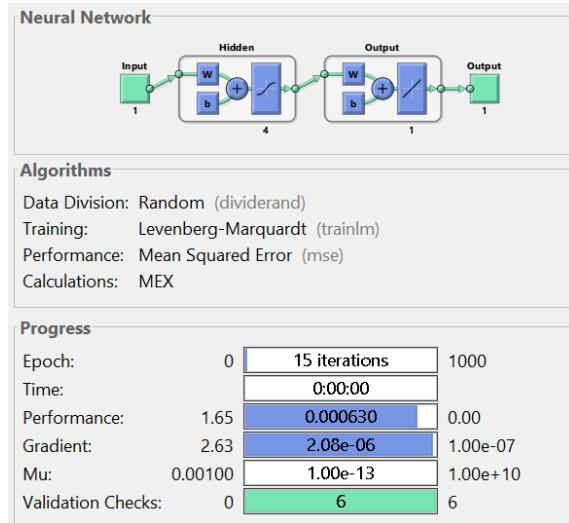
Para este método de entrenamiento se puede observar cómo el número de iteraciones, el performance y el tiempo son muy bajos.

#### Traigdm:



Este método de entrenamiento agota todas las iteraciones, además ha sido el método que más tiempo ha tardado en entrenarse (no es exagerado). La performance de este método es muy baja.

### Trainlm:



Este método de entrenamiento ha sido el que mejor resultados ha ofrecido, el tiempo, el número de iteraciones y el performance han sido los más bajos.

Como conclusión al comparar los métodos de entrenamiento, si comparamos los métodos estudiados todos los métodos ofrecen resultados muy parecidos, aunque hay uno en concreto **trainbr** que los resultados que ofrece son mucho más preciso y ha sido el único método capaz de ajustarse perfectamente a la función que queremos aproximar. En cambio, hay otros métodos que no aprenden tan bien la función, aunque le indiquemos más neuronas y por lo tanto no logran ajustarse perfectamente.

Si comparamos el rendimiento obtenido el mejor método es **trainlm**, aunque solo en cuánto a tiempos, performance y número de iteraciones, pero si observamos el resultado al aproximar la función, no es el que mejor se ha aproximado, pero ha estado muy cerca de hacerlo perfecto.

## EJ3P1

Hemos realizado la ejecución de la red propuesta sobre simplefit\_dataset obteniendo los siguientes resultados.

Podemos ver como la red creada tiene una neurona de entrada y una de salida con dos capas ocultas, la primera formada por 10 neuronas de tipo tangente hiperbólica sigmoidea (tansig) y la segunda formada por una linear (purelin).

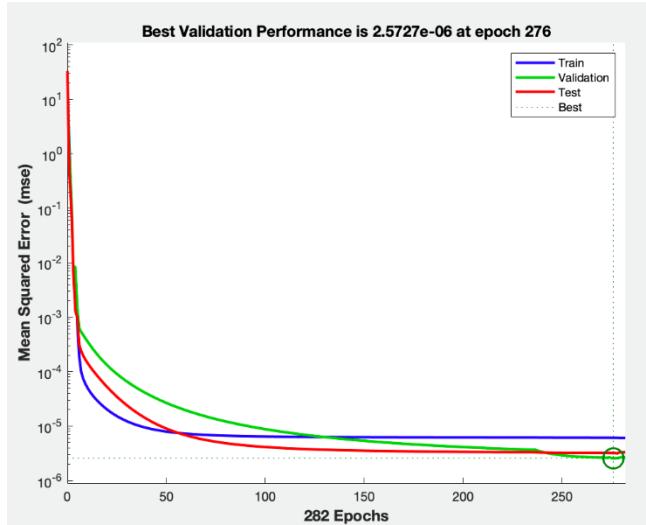
Exploraremos ahora las gráficas que se nos proporcionan para analizar el entrenamiento de la red.

### Plotperform

En esta gráfica vemos el error de cada conjunto de datos (entrenamiento, validación y test) para la red en cada época. Podemos ver como el error cae con el tiempo según avanza el entrenamiento reduciéndose para cada uno de los conjuntos de datos.

El entrenamiento para cuando se detecta un mínimo en el error de los datos de validación. Separar los datos en datos de entrenamiento y validación ayuda por tanto a distinguir entre que la red se esté aprendiendo los datos de entrenamiento y que se siga entrenando.

Por defecto la red dejará de entrenarse cuando haya obtenido consecutivamente seis incrementos en el error de los datos de validación eligiendo entonces como resultado final el mínimo producido en el error de estos.



### Plottrainstate

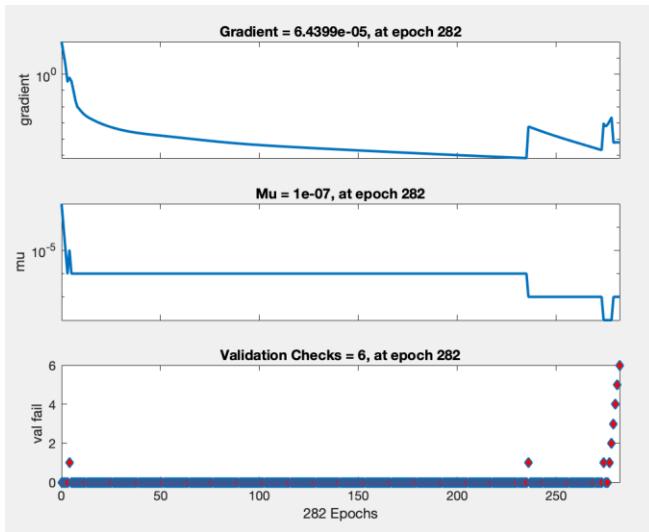
En esta gráfica veremos la evolución del entrenamiento. La gráfica se divide realmente en tres gráficas, la primera de ellas nos muestra el gradiente, la segunda la ganancia de entrenamiento y la última los incrementos de error consecutivos en los datos de validación.

Podemos ver como el entrenamiento termina al llegar a seis incrementos seguidos.

El gradiente nos informa sobre lo rápido o lento que son los cambios que se realizarán en el proceso de *backpropagation* en la siguiente iteración, este debería de descender al aproximarnos a los mínimos y ser alto al inicio del entrenamiento.

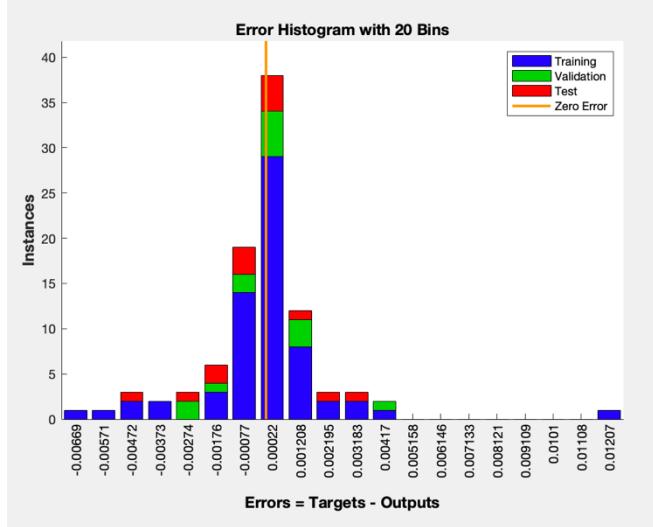
Dependiendo del método de entrenamiento que utilicemos puede que la ganancia de entrenamiento no se utilice siendo esta otro valor distinto o ninguno.

El gradiente y los errores consecutivos en los datos de validación siempre estarán presentes.



### Ploterrhist

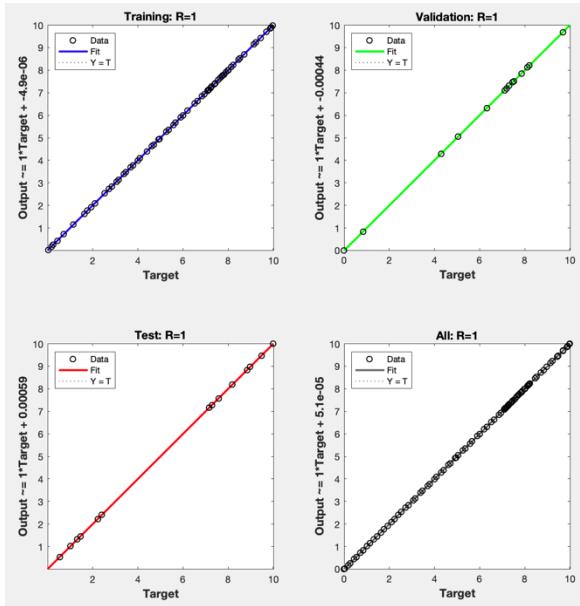
El histograma de error nos indica la cantidad de error ocurrida para cada dato de los datos de entrada de la red. En este caso vemos como la mayoría de los datos han tenido una tasa de error de 0.000222 siendo en su gran mayoría estos datos de entrenamiento (es de los que más hay). Podemos ver como se han distribuido los errores sobre nuestros datos. Podemos ver como solo unos pocos, los que están más a la derecha han sido especialmente resistentes a haber sido aprendidos por la red. En su gran mayoría para este caso concreto se han distribuido sobre errores muy bajos y de forma más o menos uniforme a lo largo de los datos de entrenamiento, test y validación.



### Plotregression

Estas gráficas nos representan el estado final de la red tras haber sido entrenada. En ellas se muestra la relación entre las salidas obtenidas y el resultado esperado sobre cada uno de los conjuntos de datos que manejamos.

En un entrenamiento perfecto la gráfica mostrará una línea recta donde la salida producida por la red es siempre la salida esperada. Cuando el entrenamiento no haya sido perfecto (un caso realista) las salidas obtenidas no se corresponderán con las esperadas en todos los casos, deberán aproximarse a ellos. La inclinación y posición de la recta de regresión obtenida sobre los puntos de la gráfica representará la desviación media entre los resultados obtenidos y los esperados.

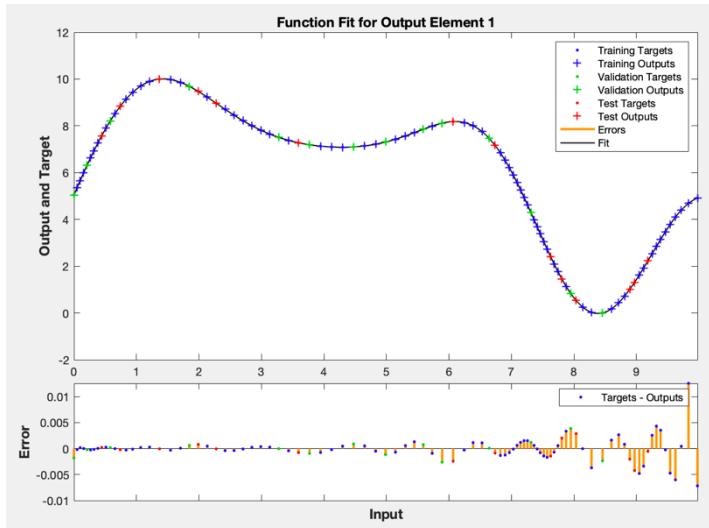


### Plotfit

Nos genera una representación visual de la función a la cual la salida de la red se ajusta. Sobre esa función y en líneas verticales se nos representará el error de cada punto de la función respecto del que debería de ser.

Cuanto más se aproxime la función generada respecto a la función ideal que representan los resultados esperados mejor habrá sido el entrenamiento y menor será el error.

Esta función solo se podrá generar cuando solo haya una neurona de entrada.



### bodyfat\_dataset

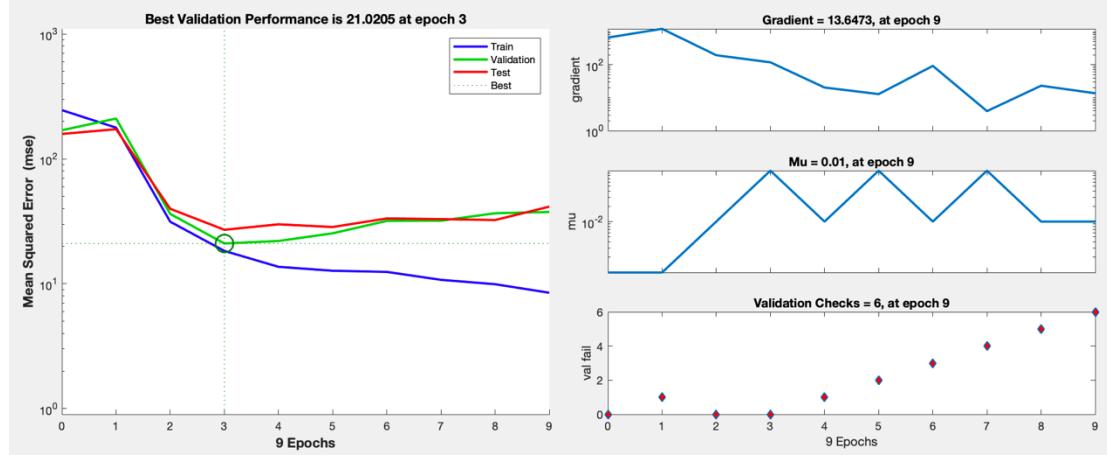
Analizaremos a continuación unos datos más realistas como los son los de bodyfat\_dataset donde la red no debería ser capaz de obtener unos resultados perfectos teniendo por tanto margen de error en sus predicciones.

En este caso debido al tipo de datos utilizado el cual tiene 13 parámetros como datos medidos esperando inferir de ellos una respuesta de positiva o negativa tendremos 13 neuronas de entrada y 1 de salida.

Podemos ver que no se han realizado todas las iteraciones que podrían haberse hecho de modo que los errores de los datos de validación han cortado el aprendizaje con prontitud.

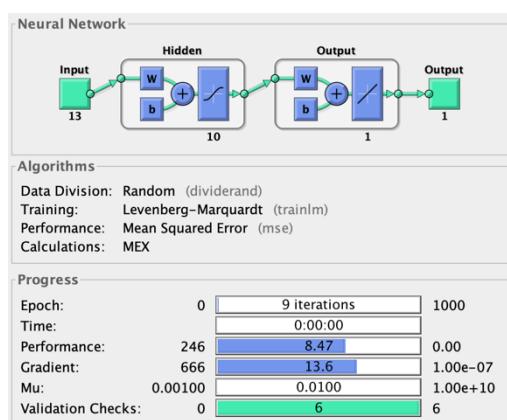
Podemos ver como en la primera época la red no ha sido capaz de aprender nada de los datos de entrada de modo que su error ha aumentado. En respuesta la red ha aumentado el learning

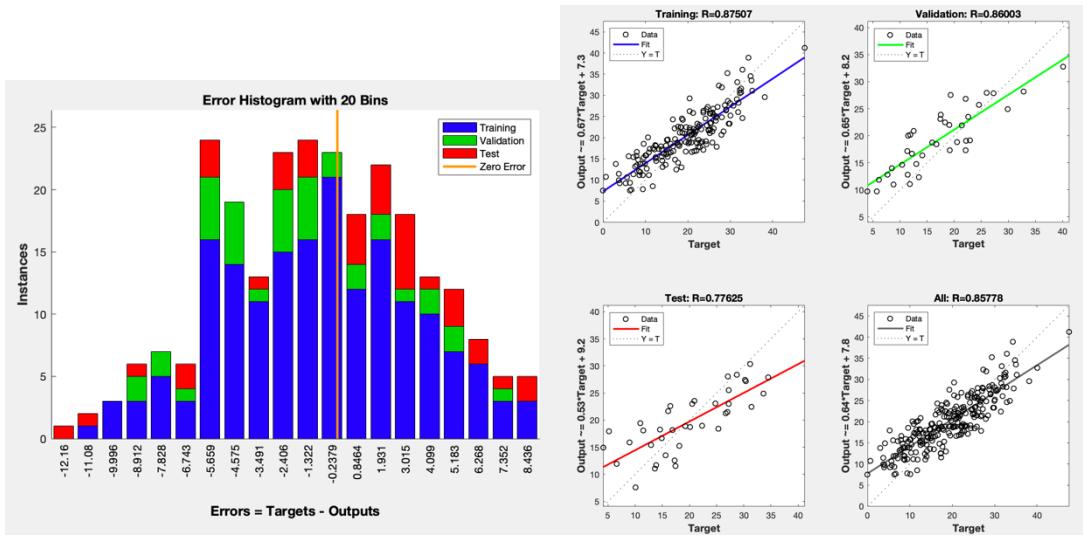
rate ( $\mu$ ) logrando en este punto aprender de los datos. Hasta la época 3 se sigue aprendiendo, pero a partir de ella a pesar de que el error en los datos de entrenamiento cae en los datos de validación aumenta. En este punto la red está aprendiéndose los datos de entrenamiento, pero no está logrando inferir información extra de ellos. Cuando varios aumentos del error se suceden la red comienza a modificar alternativamente  $\mu$ . Finalmente, cuando seis épocas seguidas se suceden sin dejar de aumentar el error en los datos de validación el entrenamiento finaliza.



Vemos en el histograma como los errores son ahora mayores que en el caso anterior. También podemos apreciar como los errores en los datos de validación, test y entrenamiento están distribuidos por igual, es decir, ninguno de los sets es más propenso que el resto a acumular errores, la red puede inferir información.

En las rectas de regresión vemos como el entrenamiento no ha sido perfecto. Los mejores resultados se obtienen con los datos de entrenamiento y los peores con los de test. No obstante, podemos apreciar como sí hay relación entre las salidas obtenidas y las esperadas, es decir, la red ha aprendido.





### Modificación de la división de los datos de entrenamiento

Realizamos cambios en la división de los datos de entrenamiento y exponemos los resultados obtenidos con cada una de las divisiones junto a una explicación de por qué se producen dichos resultados.

- Si establecemos que todos los datos sean de entrenamiento la red entrenará hasta agotar sus iteraciones pues no tiene los datos de test para saber cuando parar de entrenar. Se obtienen muy buenos resultados (performance muy baja) pues la red a aprendido los datos lo cual hace que pierda su capacidad de inferir información sobre datos nuevos.
- Las combinaciones con pocos datos de test no logran bajar el error de los datos de validación y terminan pronto con resultados muy variables.
- Si establecemos los datos de validación como demasiado bajos la red terminará o muy pronto con muy malos resultados o muy tarde con unos resultados en la media de los esperado, depende de cuanto se parezcan los pocos datos de validación a los de entrenamiento.
- Si los datos de test son demasiado pocos los resultados podrán ser demasiado buenos o demasiado malos, esto podría falsear resultados si la ejecución resulta haber sido favorable respecto de la distribución de los datos.

En general parece ser recomendable que haya la misma cantidad de datos de validación que de test y que siempre haya más datos de entrenamiento que del resto, aunque sin que de los otros haya demasiado pocos. Es por eso que 70, 15, 15 parece una distribución tan ideal.

### Cambio en el método de entrenamiento

Se ha evaluado la performance obtenida y el número de iteraciones obteniendo que las funciones trainlm, trainbr y trainscg eran las que proporcionaban mejores resultados de forma más consistente, menor media y desviación. Por el contrario, traingd, traingdm y traingdx proporcionaban los peores, mayor media y desviación. Resulta destacable que cuando los resultados eran peores también eran menos consistentes. Resulta destacable también que las funciones con peores resultados sean las que utilizan el algoritmo de descendimiento del gradiente de forma más pura.

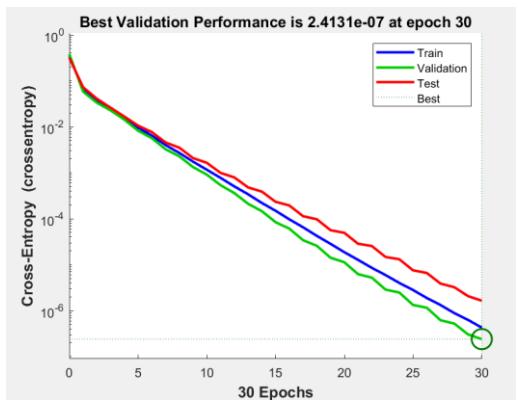
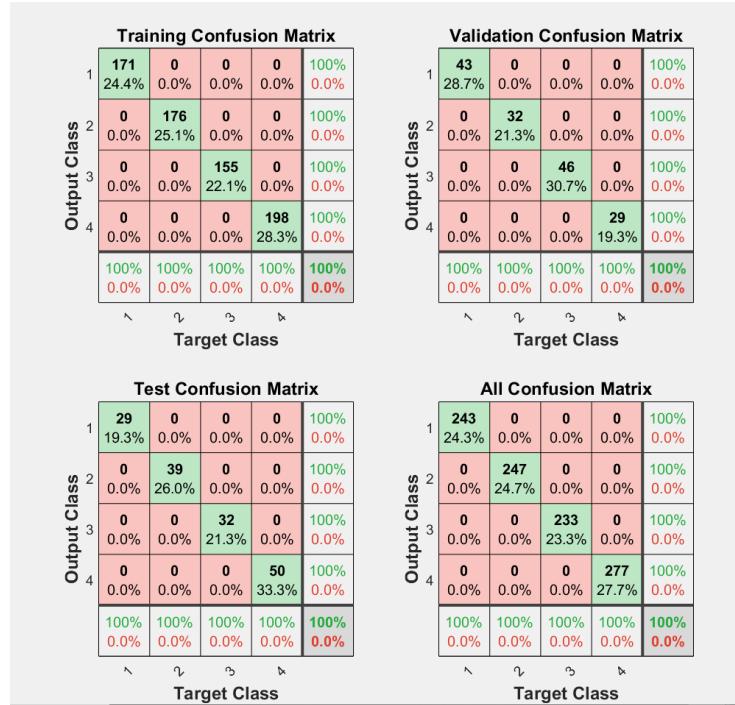
Las funciones trainbr y trainbfg han destacado por producir de media muchas más iteraciones que el resto de las funciones.

Por lo general se observa que el resto de las funciones proporcionan resultados similares a las que han proporcionado los mejores sin demasiada distinción entre ellos para el conjunto de datos de bodyfat\_dataset.

## EJ4P1

En este ejercicio primero vamos a realizar la clasificación con los datos, mediante la red llamada *patternet*, y con el conjunto de datos *simpleclass\_dataset*, para ver el funcionamiento. Luego cambiaremos el método de entrenamiento y el conjunto de datos para estudiar el comportamiento de esta red variando estos parámetros.

Para el ejemplo con los datos de entrada *simpleclass* y con la división 70/15/15 se puede ver que no ha dado ningún error a la hora de predecir las clases dándole los inputs.



Y además la gráfica del error muestra como no para de descender hasta que se considera que ha llegado al mínimo, ósea que la red no paraba de aprender. Por estos motivos se observa que los resultados obtenidos son los mejores que se podían obtener.

Ahora vamos a realizar con el conjunto de datos *cancer\_dataset* y también variando los métodos de entrenamiento y las divisiones de los datos.

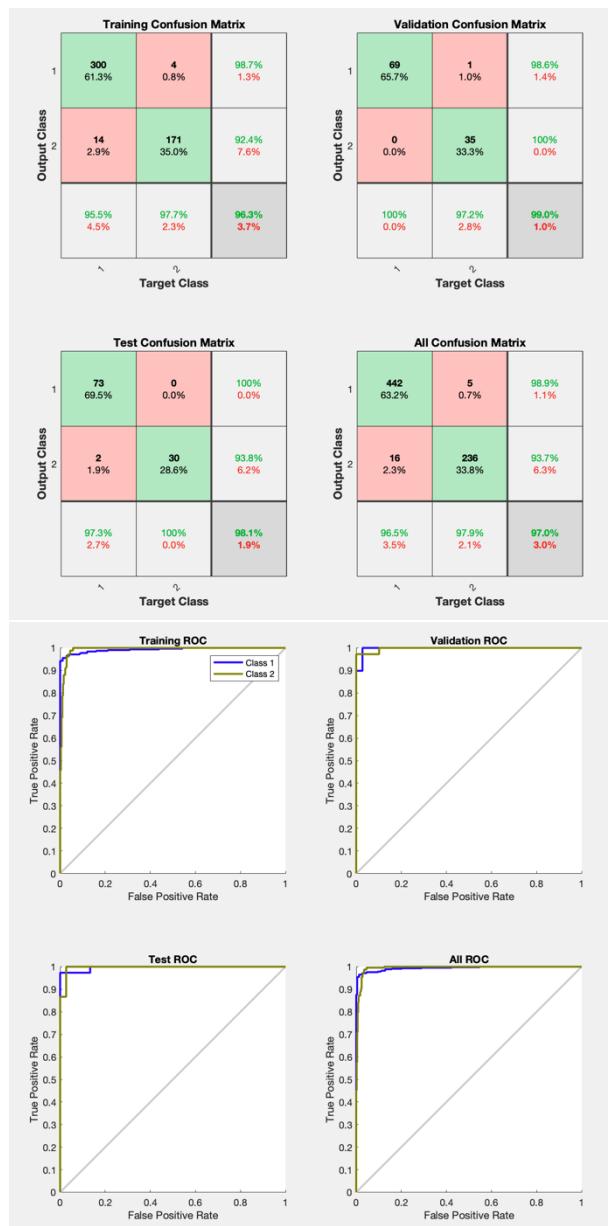
Para el conjunto de entrenamiento cancer, con el método trainbr, separando los datos de esta forma:

70% - Entrenamiento

15% - Validación

15% - Test

Se observa que, aunque la clasificación es bastante buena, existen pocos datos que estén mal clasificados, solo un 3%. Esto es debido a que el conjunto de datos es más complejo y por lo tanto es más complicado a la hora de encontrar patrones. Aunque en este caso han sido pocos los errores, para lo que estamos estudiando son demasiados ya que al ser algo tan sensible como el cáncer no se pueden cometer ningún tipo de errores.



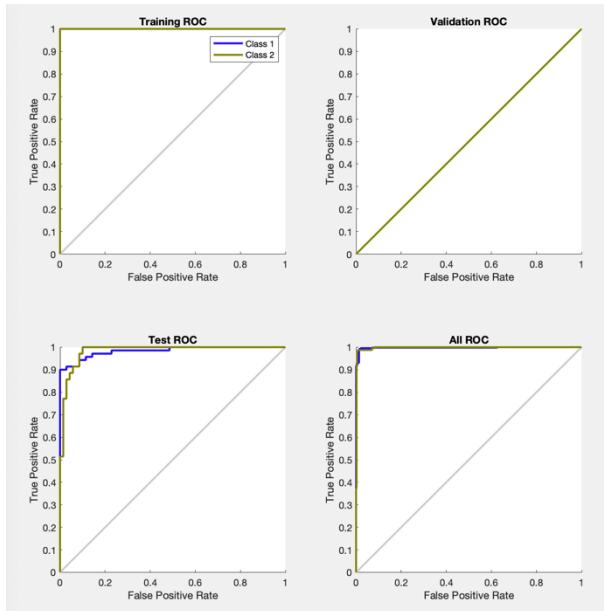
La gráfica de ROC relaciona el ratio de los verdaderos positivos con los falsos positivos al entrenar la red para los datos de train, validación y test. En este caso se aprecia como son mucho mayor los verdaderos positivos que los falsos positivos.

Se procede a cambiar el método de entrenamiento y comparar cuáles dan mejores resultados, es decir, menor porcentaje de error. Los métodos que se van a estudiar son los siguientes, manteniendo la división de los datos:

- **Trainrp:**

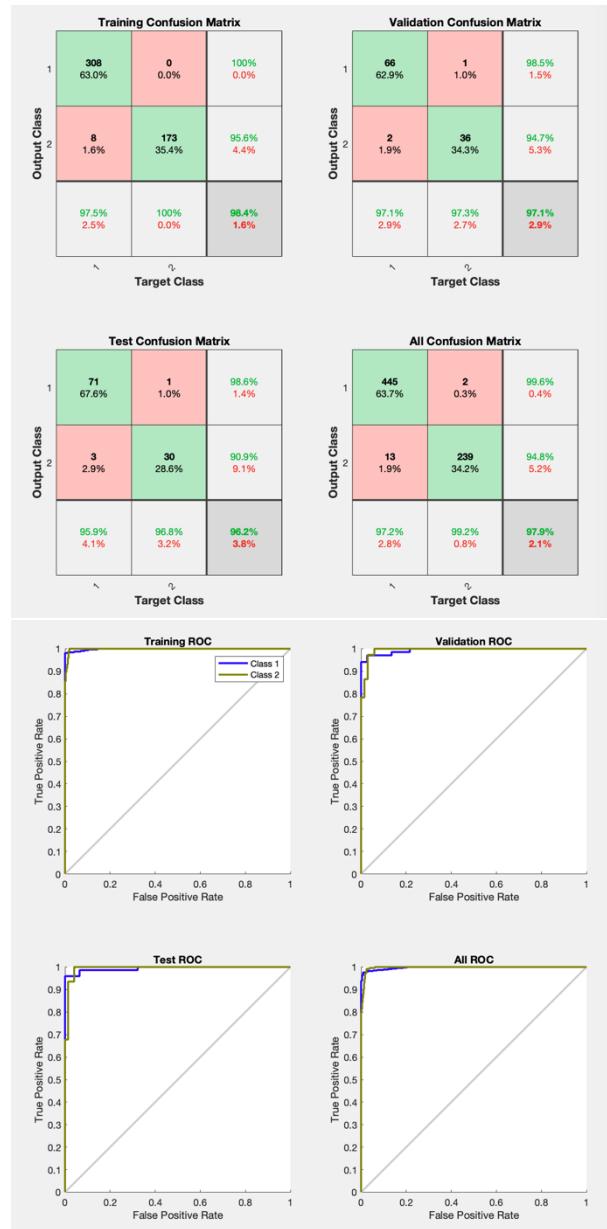
Este método lo he comentado anteriormente.

- **Trainbr:**



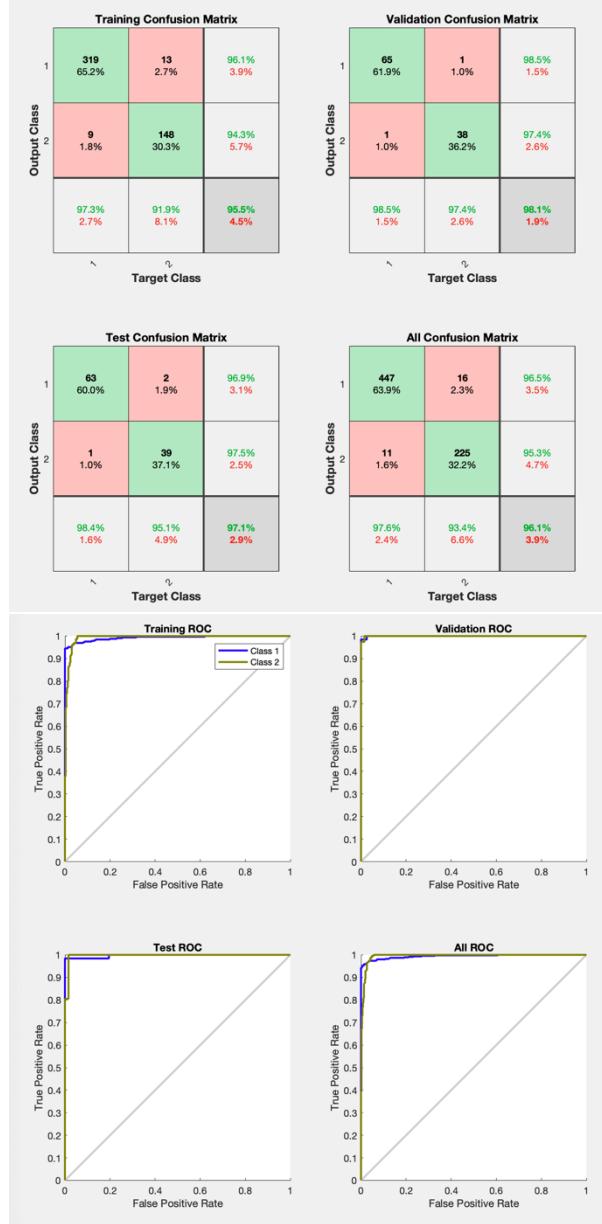
En general este método de entrenamiento ofrece menos error en los datos de entrenamiento que el anterior, pero en los datos de test el error es muchísimo mayor que el método anterior. Además, nos hemos dado cuenta de que la matriz de confusión para los datos de validación no la muestra.

- **Trainlm:**



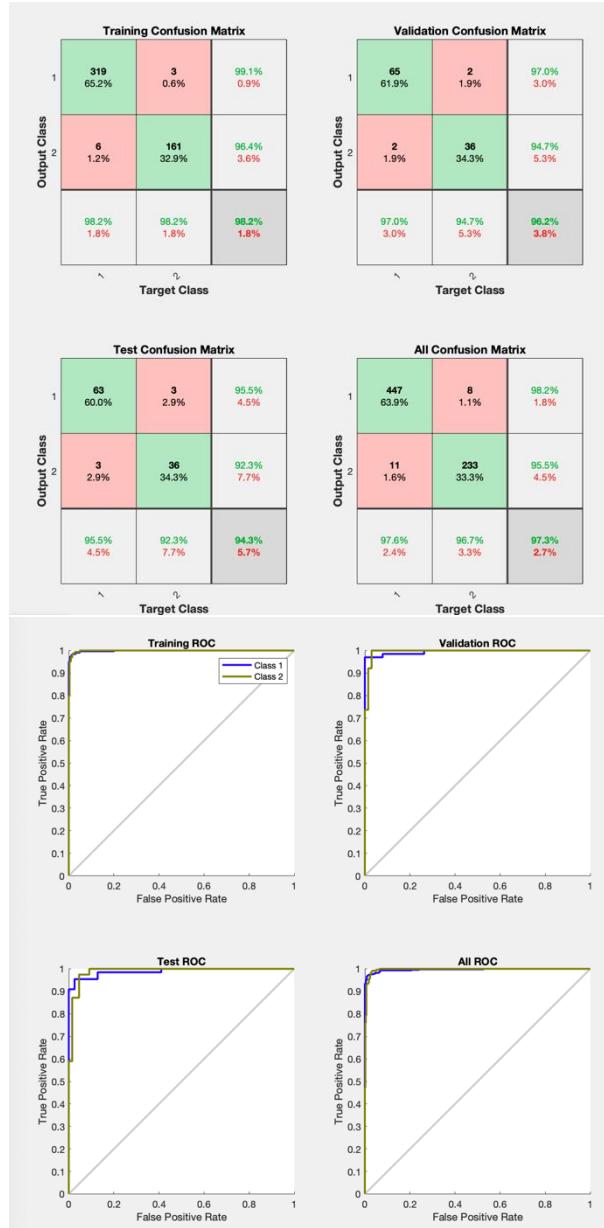
En este método los errores de los 3 en general son bastante buenos y muy parecidos a los del primer método, los falsos positivos y los falsos negativos que predice son muy bajos. El error total es 2,1% y el error para los datos de test es 1,6%.

- **Trainbfg:**



Este método de entrenamiento ofrece peores resultados que los anteriores, aunque tampoco son realmente malos. Pero se puede observar como los porcentajes de errores para los datos de validación es menor, así que en teoría ofrece mejores resultados, al menos en este caso.

- **Traincgb:**



Para este método de entrenamiento, los resultados buenos excepto para los datos de test, esto significa que la red no se ha entrenado bien y por lo tanto los resultados no son buenos.

Podemos concluir que para este conjunto de datos y para esta división de datos concreta el método que mejor rendimiento ofrece y que mejor clasifica las muestras es el de *trainrp* junto al de *trainbfg*.

Ahora vamos a cambiar la división de los datos utilizando el método de entrenamiento que mejor resultados ha dado, *trainrp*.

Las divisiones de los datos que vamos a probar son las siguientes:

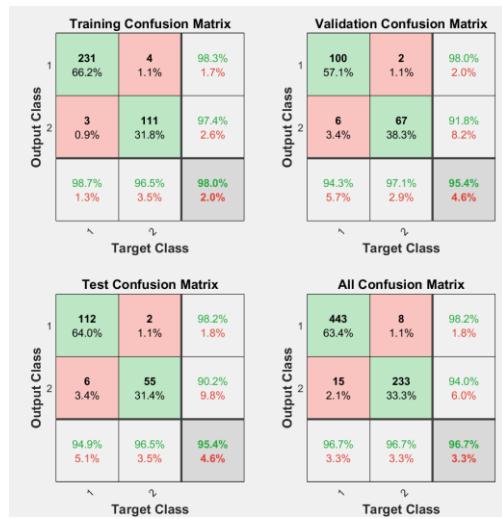
**1º:**

**50% - Entrenamiento**

**25% - Validación**

**25% - Test**

Al realizar esta división de los datos se observa que en general el error aumenta, aunque tampoco excesivamente y esto es debido a que la cantidad de datos que tiene para entrenar la red es la misma (muy pocos datos) respecto a los datos que tiene para comprobar si predice correctamente.



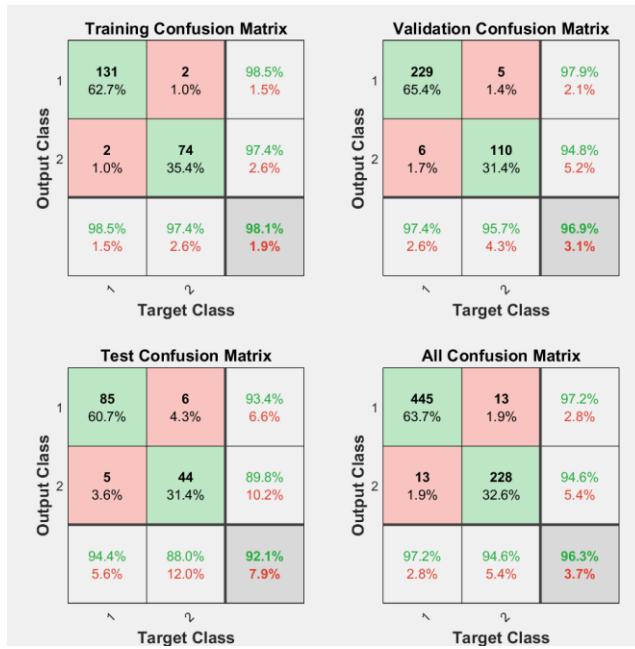
**2º**

**30% - Entrenamiento**

**50% - Validación**

**20% - Test**

Para esta división de datos se observa como los errores son muy parecidos a la división anterior, excepto para los datos de test. Esto es debido a que como hemos anteriormente, los datos que tiene para entrenar a la red son muy pocos y el entrenamiento no llega a ser completo ni óptimo.



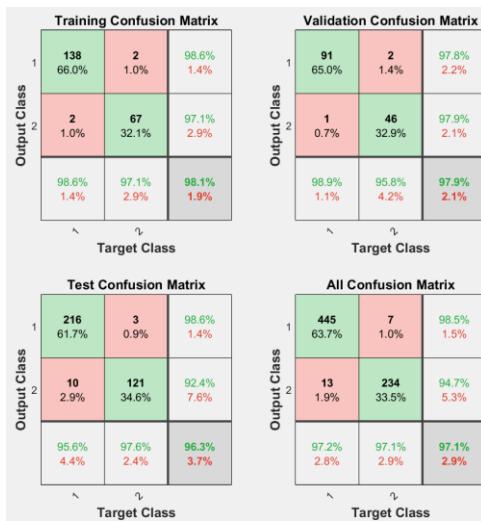
**3º**

**30% - Entrenamiento**

**20% - Validación**

**50% - Test**

Con esta última división se observa que los errores obtenidos para los datos de test son muy grandes en comparación con los errores de los datos de entrenamiento y los datos de validación. Ha salido este resultado porque como los datos de entrenamiento son muy pocos, la red no ha sido capaz de encontrar más patrones y así poder utilizarlos para clasificar los datos de test y los de validación.

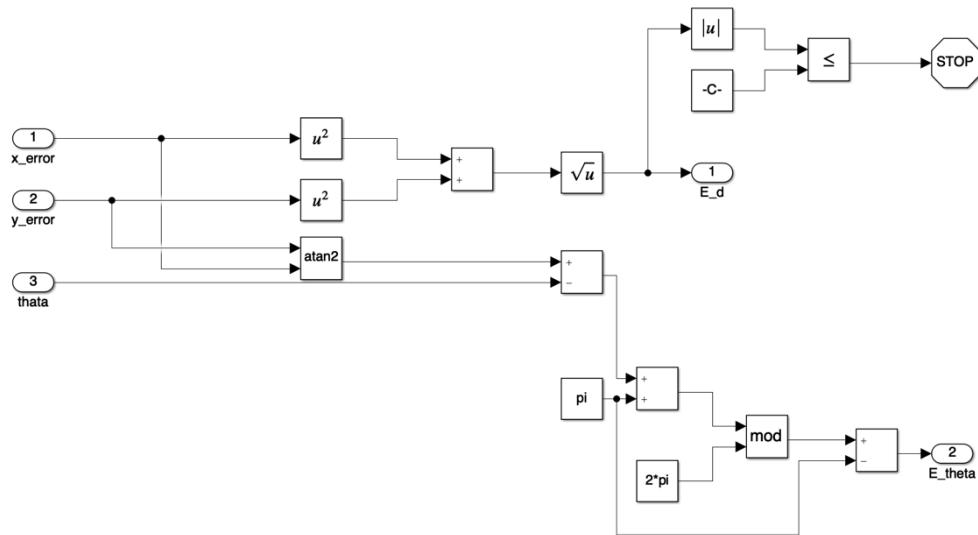


Después de realizar este estudio podemos concluir que la mejor forma de dividir los datos es de 70% para entrenamiento, 15% para validación y 15% para test. Esto es así porque al tener una gran cantidad de datos para entrenar, la red puede identificar los patrones de más datos y así luego aplicarlos a los datos de test y los datos de validación y generalizar los resultados de forma correcta.

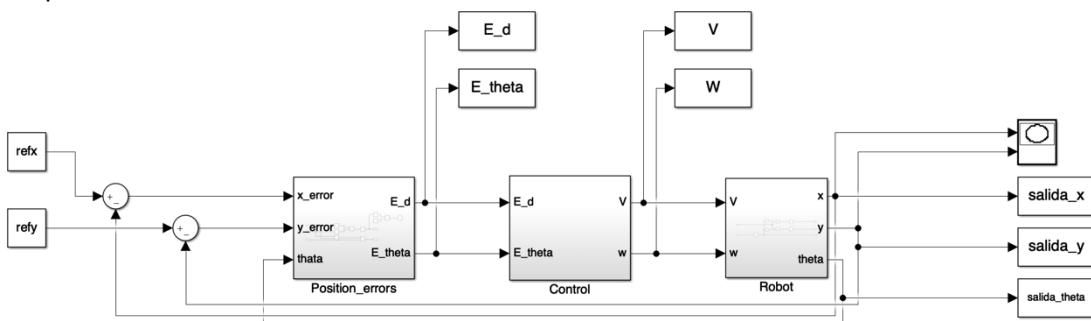
## EJ1P2

Creamos en Simulink el módulo que calcula los errores que harán de entrada del módulo de control, el cual trataremos como una caja negra. Junto al cálculo de los errores ubicamos también la parada de la simulación que sucederá cuando estemos por debajo de cierto umbral de distancia de nuestro destino.

Para comprobar que la salida del error de theta estuviera entre los límites de  $[\pi, -\pi]$ , se ha sumado  $\pi$  para que el rango quedase entre  $[0, 2\pi]$ , en el caso de que al sumarle  $\pi$  se pasase del rango, por ejemplo  $3\pi$  se realiza un módulo para que este entre el rango de  $[0, 2\pi]$ . Luego se le resta  $\pi$  para asegurarnos de que se encuentra en el rango deseado,  $[-\pi, \pi]$ .



Posteriormente enlazamos el módulo de cálculo de errores con el de control y finalmente con la planta del robot. Creamos variables de entrada y salida para poder controlar y observar el comportamiento de todo el sistema desde Matlab.



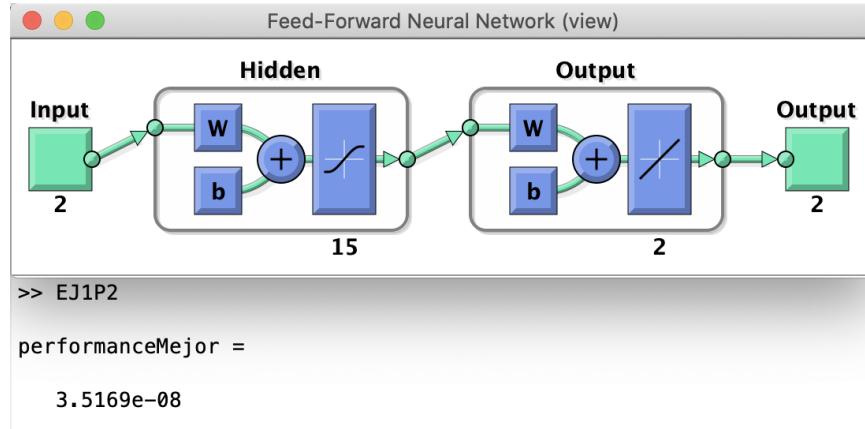
Desde Matlab generamos valores que aplicamos a refx y refy para controlar el robot y capturamos los de las velocidades V y W y también los de las entradas del módulo de control  $E_d$  y  $E_\theta$ . Es decir, realizamos una simulación en la que al robot le proporcionamos objetivo y capturamos como se comporta el módulo de control para llegar a él ante cada entrada de las que se le proporciona.

A partir de los pares entrada-salida obtenidos entrenamos una red neuronal con el objetivo de luego sustituir el módulo de control por ella.

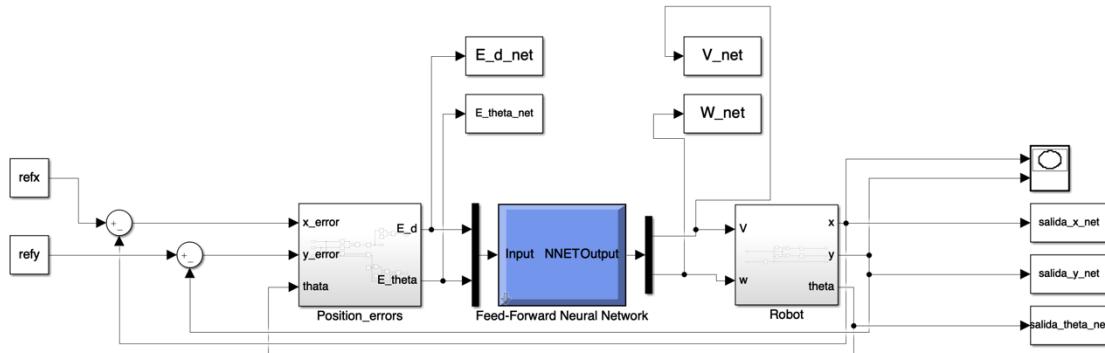
Para crear la red neuronal hemos probado con diferente número de neurona, del rango de 5 a 25 y así elegir la que mejor performance (poco performance) y la que menor error tenga.

Para ello hemos realizado un bucle que vaya probando con todas las neuronas y solo guardamos la que tenga mejor performance.

En este caso el mejor resultado ha sido con 15 neuronas con un performance de 3.1569e-08.

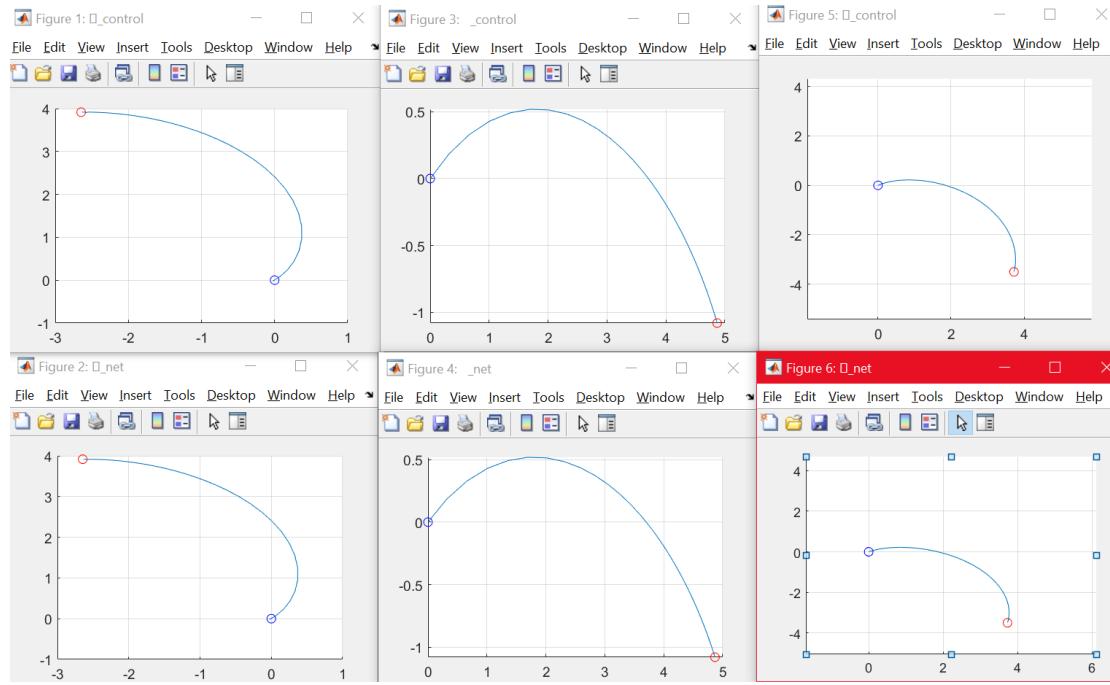


Una vez que hemos generado la red neuronal creamos ahora un módulo de Simulink con ella.

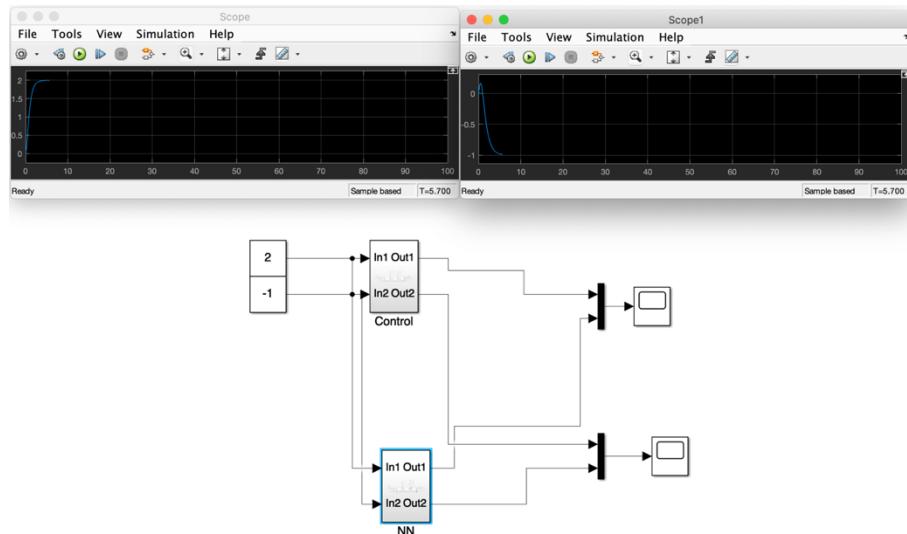


Proporcionamos ahora las mismas entradas a ambos módulos y observamos como cada uno se comporta ante ellas obteniendo las siguientes observaciones.

- Visualmente las trayectorias producidas por ambos módulos son muy similares.
- Vemos que cada módulo proporciona salidas muy parecidas o incluso iguales tanto en valor como en cantidad, se llega a la vez al objetivo y se realiza la misma trayectoria.



En el comparacionEJ1P2.M hemos creado el código para compararlo de forma automática.



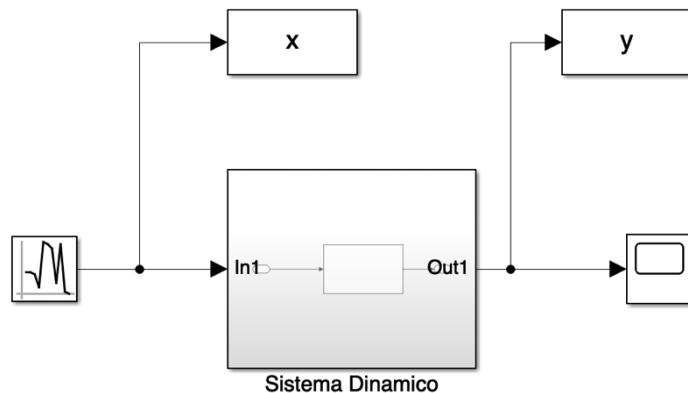
También hemos utilizado un scope para comparar las salidas que ofrece el robot con el controlador y utilizando la red neuronal, para ello utilizamos 2 scope uno para comparar las X y otro para comparar las Y. Como se observa en los gráficos los resultados son exactos, no existe ningún tipo de error, es decir, los valores están superpuestos respectivamente para cada gráfico (el de la derecha representa las X del controlador y de la red neuronal y el de la izquierda representa las Y del controlador y de la red neuronal).

## EJ1P3

Para este primer ejercicio hemos copiado el esquema que se indicaba en el pdf, para posteriormente ejecutarlo y guardar los resultados obtenidos. El propósito de este ejercicio es crear y entrenar una red dinámica (un tipo de red neuronal que a la entrada le llega la salida retardada en el tiempo) para simular el sistema dinámico.

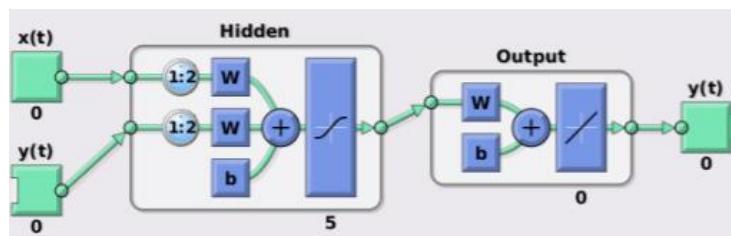
El sistema que se ha creado para probarlo se ha llamado `test_bench`. Se ha configurado tal cuál decía en el enunciado de la práctica,  $T_s = 0.1$ , Solver con tiempo discreto, variables  $x$  e  $y$  accesibles desde el workspace con estructura “Structure with time”.

Al esquema se le ha añadido una caja negra que es el bloque sistema dinámico.



Una vez creado el sistema se procede a ejecutarlo para guardar los resultados de entrada que es la variable X y de salida que es la variable Y.

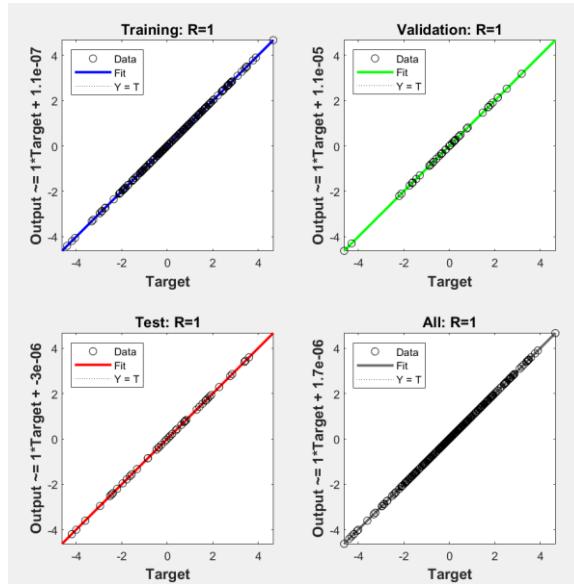
Se procede a generar una red NARX con 5 neuronas y dos retardos en la entrada y en la salida realimentada.



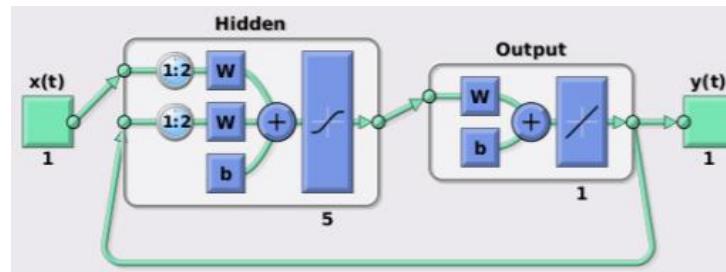
**Muy importante:** Para poder entrenar la red hay que hacerlo sin que este cerrada, nosotros sabemos los resultados que la red da y por lo tanto podemos retardarlos para meterlos en el input en el momento justo. Una vez que la red esta correctamente entrenada se debe cerrar el lazo y conectar la salida con la entrada y los retardadores (se va a explicar más adelante).

Una vez generada la red, se entrena con los datos obtenidos de la simulación, pero para ello hay que transformar las salidas y las entradas.

Al entrenar la red se puede observar como ofrece unos resultados muy buenos, en el gráfico de regresión se ve como predice correctamente todos los valores y por lo tanto forma una línea recta como debería salir (con todos los puntos dentro de esa línea).



Una vez entrenada la red con las salidas y las entradas, se procede a cerrar el lazo que une la salida con las entradas para poder utilizar la red normalmente.

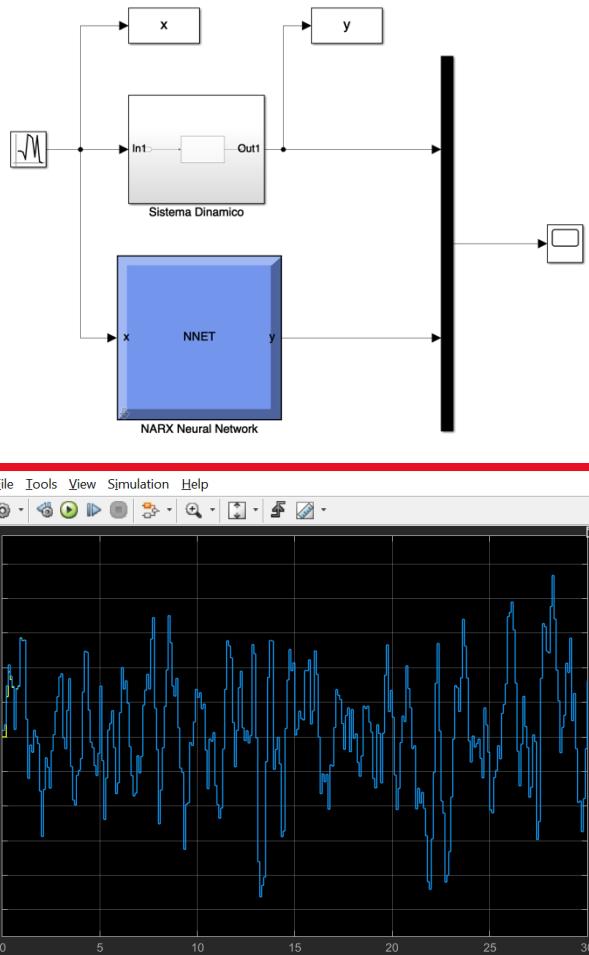


Si observamos el rendimiento de la red, en general es muy bueno ya que la performance conseguida es muy baja. En cambio ha tenido que utilizar el máximo número de iteraciones y por lo tanto ha tardado un poco más al entrenarse.

Progress					
Epoch:	0	1000 iterations	1000		
Time:		0:00:04			
Performance:	8.29	1.42e-09	0.00		
Gradient:	18.7	1.28e-05	1.00e-07		
Mu:	0.00100	1.00e-07	1.00e+10		
Validation Checks:	0	0	6		

Después de cerrar el lazo y tener la red entrenada se procede a crear una caja negra en simulink para poder usarla como un bloque y conectarla en el sistema anterior.

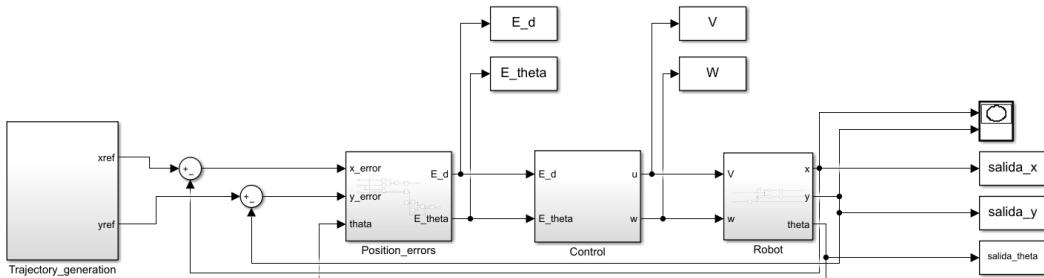
Una vez obtenido el esquema anterior, se procede a realizar la simulación multiplexando la salida de la red y la salida del sistema para después mostrarlo en scope.



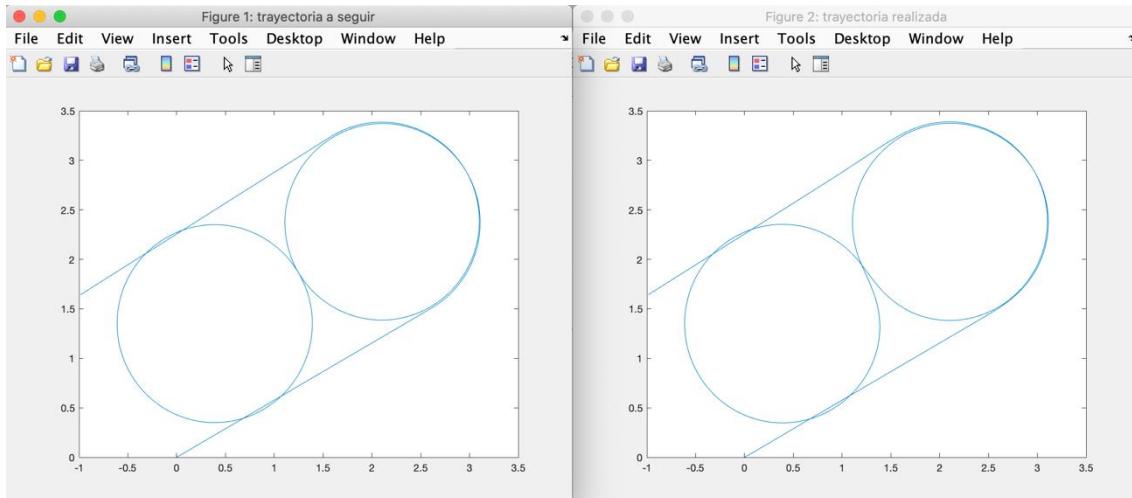
Como se puede observar en la gráfica del scope de dos canales, se muestra que la red neuronal es parecida casi a la perfección la simulación, ha sido capaz de generalizar los datos de entrenamiento de forma correcta. El único error que ha tenido ha sido al principio, que podemos verlo en amarillo y ha sido relativamente muy poco.

## EJ2P3

Para este ejercicio hemos creado en simulink el modelo de abajo, configurando todos los parámetros igual que las sesiones anteriores.  $Ts = 100e-3$ ,  $\theta_0 = 30\pi/180$ ,  $x_0 = 0$  y  $y_0 = 0$ . Con un solver de tipo fixed-step y configurado como ode3.



Hemos realizado un script que ejecute los dos sistemas, el de la izquierda es únicamente la caja Trajectory\_generation y el de la derecha es el de la trayectoria realizada por el robot después de aplicarle el controlador.



En este apartado vamos a crear una red de tipo narx, para ello utilizamos los datos generados al utilizar el controlador anterior.

Creamos la estructura de la red con 1 capa oculta, hemos ido probando con diferentes números de neuronas y a la vez viendo el resultado que ofrecía al aplicar la trayectoria sobre la red generada.

La red tiene 1 retenedor para el feedback la realimentación de la salida y dos retenedores para la entrada de los datos.

```

net = narxnet(1:2, 1, [i]);
[x,xi,ai,t] = preparets(net,inputsc,{},outputsc);
net = train(net,x,t,xi,ai);
    
```

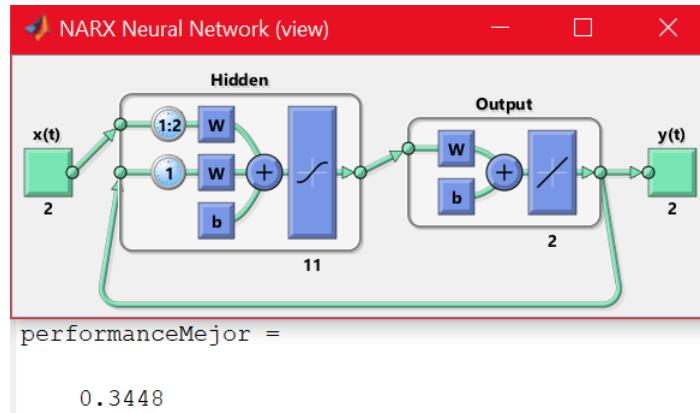
Después de entrenar la red para poder probarla y ejecutarla con otros datos es necesario cerrar el lazo, para que la realimentación se obtenga directamente de la salida de la red y no haya que “preparar los datos” para que funcione.

```
net = closeloop(net);
```

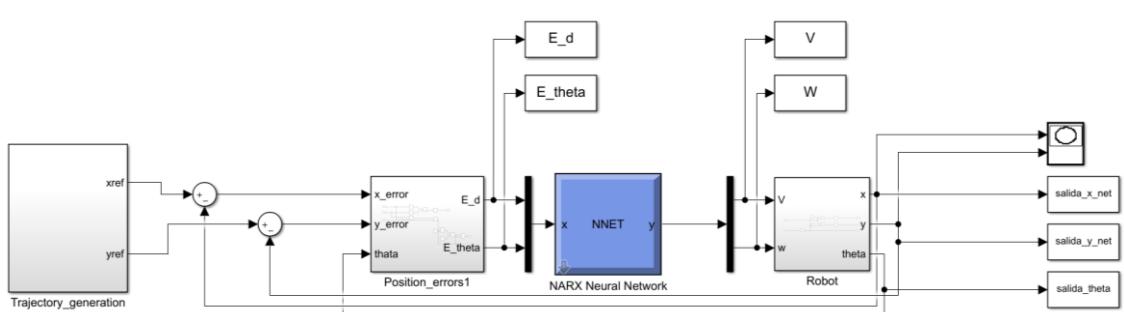
Para entrenar la red, hemos cogido los siguientes datos del control:

$$\begin{aligned}x_0 &= 0, y_0 = 0 \\x_0 &= 0, y_0 = 1 \\x_0 &= 1, y_0 = 0 \\x_0 &= 1, y_0 = 1\end{aligned}$$

Con el entrenamiento anteriormente comentado hemos obtenido que la mejor red ha sido con 11 neuronas en la capa oculta con un performance de 0.3448, hemos realizado pruebas de 5 a 20 neuronas, y la que mejor resultados nos ha dado al probar la red ha sido la de 11 neuronas como hemos comentado.



Después de obtener la red, mediante el comando gensim, obtenemos un bloque con la red para poder utilizarlo en las simulaciones con simulink, después de obtener el bloque cambiamos en el sistema anteriormente creado el controlador por la red, con un multiplexor para combinar las entradas y con un demultiplexor para separar las salidas.

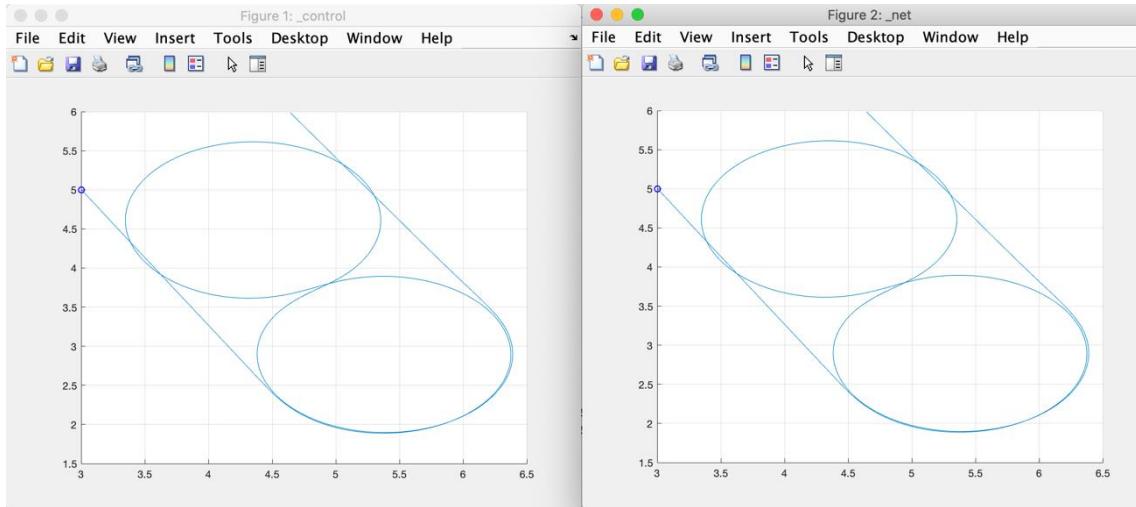


Después de crear la red, procedemos a probarla y simular el comportamiento con diferentes puntos de entrada, para ello creamos a un script para compararlo de forma más cómoda.

En el script **ComparacionEJ2P3.m** lo hemos creado variando la **x\_0** e **y\_0** desde 1 hasta 3, para ver todas las posibles combinaciones y como es el resultado:



Vemos que si elegimos un ángulo y posición inicial radicalmente distintos a los de entrenamiento la red es capaz de inferir el control que debe realizar sobre la planta de modo que esta acaba produciendo la trayectoria deseada.



Como podemos comprobar la red neuronal ha sido capaz de realizar igual que el controlador todas las pruebas. Por lo que podemos concluir que la red NARX creada funciona de forma correcta.