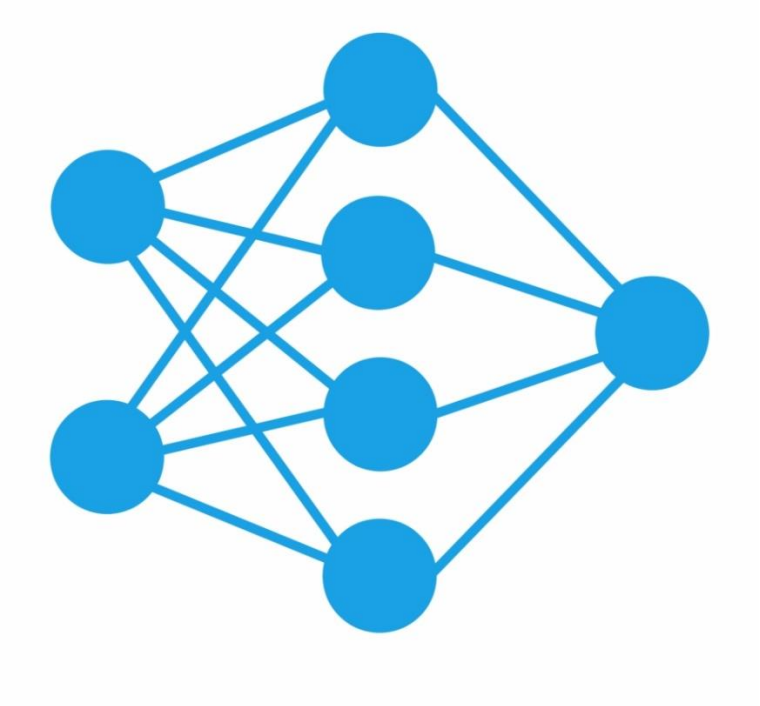
Práctica 1: Identificación y control neuronal



SISTEMAS DE CONTROL INTELIGENTE

05/10/2019

Autores: Álvaro de las Heras Fernández
Álvaro Maestre Santa

Índice

1.Introducción

El objetivo de realizar esta práctica consiste en realizar una serie de simulaciones de robots a los que se aplicarán control clásico y neurocontroladores, realizando comparaciones entre ambos y realizando variaciones a los controladores neuronales, para observar cómo afecta a la trayectoria. Toda esto se realizará en tres partes, siendo la primera una como introducción a las redes neuronales, la segunda una comparación entre controles y la tercera para seguir trayectorias preconfiguradas.

2.Desarrollo de la práctica

Parte I

Ejercicio 1

En este ejercicio se ha realizado un clasificador neuronal mediante un perceptrón. Para ello se dispondrán de puntos con 4 tipos de clases que son los siguientes:

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

Fig. 1 Datos que se clasificarán.

Una vez introducimos los datos en Matlab, se ha de crear y entrenar el perceptrón que simplemente se ha de llamar por su nombre, y después entrenarle con train() al que introduciremos los puntos y clases, estas últimas como objetivos.

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

La red con los datos que recibe por entrada es capaz de separar los datos en sus 4 clases mediante las 2 rectas delimitando con ellas 4 áreas. Todo este proceso lo hace el perceptrón iterando 8 veces (probando distintas posiciones con las rectas), hasta conseguir tener los puntos clasificados.

La capa de salida tiene 2 neuronas como indica la Fig. 2, esto se debe a la cantidad de clases que hay que clasificar. En este caso con al tener cuatro clases como mínimo necesitábamos 2 salidas para que se puedan asignar a las clases cuando se clasifique y entrene la red [00=Clase 0 01=Clase 1 10=Clase 2 11=Clase 3], por lo que eran necesarias 2 neuronas a la salida. Si únicamente hubiéramos tenido que clasificar en dos clases con una única neurona en la capa de salida hubiera sido suficiente.

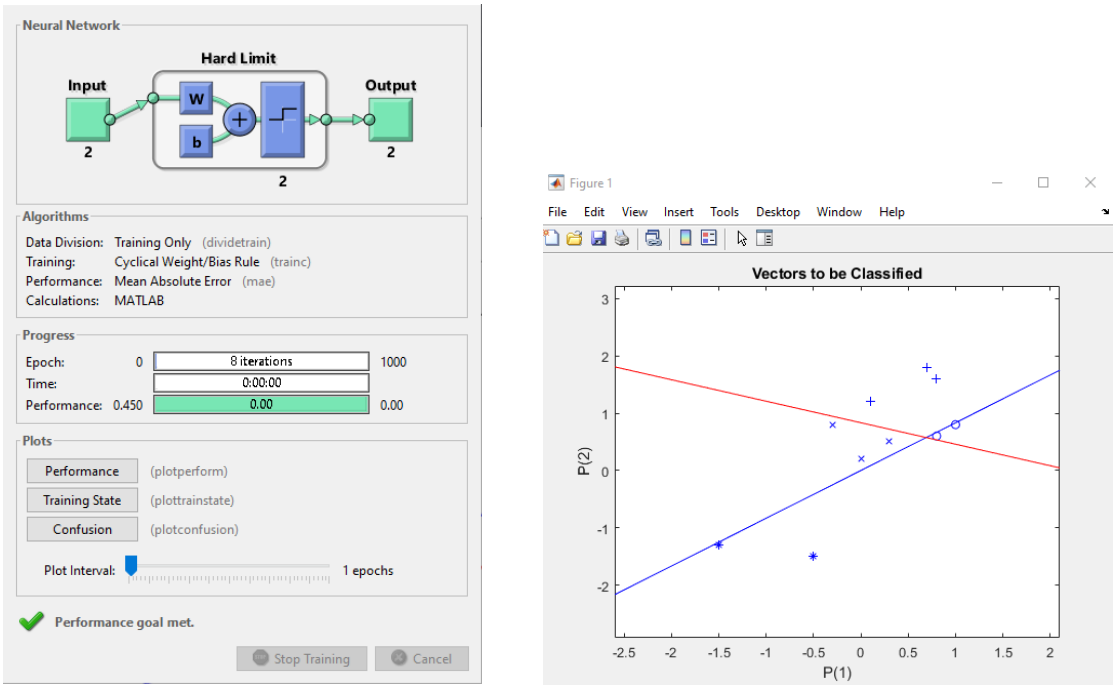


Fig. 2 Salida del entrenamiento y representación del resultado, respectivamente.

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

Al incorporar ese nuevo punto ya no se puede clasificar, haciendo que el perceptrón itere indefinidamente, llegando al límite máximo de 1000 iteraciones. La razón de que ahora no pueda clasificarlo se debe a uno de los principales inconvenientes del perceptrón es que cuando se tiene un problema que no es linealmente separable, no puede converger porque no tiene solución haciendo que busqué una solución indefinidamente.

Ejercicio 2

En este ejercicio te pide que cambies el tipo de entrenamiento de las redes y los compares, para ello hemos utilizado 4 tipos de entrenamientos, que son:

- Trainrp: Es una función de entrenamiento de red que actualiza los valores de peso y de desviación de acuerdo con el algoritmo de resilent backpropagation (Rprop).
- Trainlm: Es una función de entrenamiento de red que actualiza los valores de peso y de desviación de acuerdo con la optimización Levenberg-Marquardt. A menudo es el algoritmo de backpropagation más rápido, como podremos ver en este ejercicio.
- Trainbr: Es una función de entrenamiento de red que actualiza los valores de peso y de desviación de acuerdo con la optimización Levenberg-Marquardt. Minimiza una combinación de errores y pesos al cuadrado y, a continuación, determina la combinación correcta para producir una red perfecta. Este proceso se llama regularización bayesiana.
- Traingd: Es una función de entrenamiento que actualiza los valores de peso y de desviación según el descenso de gradiente.

A parte hemos modificado también las neuronas para ver cómo se comporta con más y menos neuronas. Los resultados son los siguientes:

- Trainrp(4 neuronas): Resilient Backpropagation.

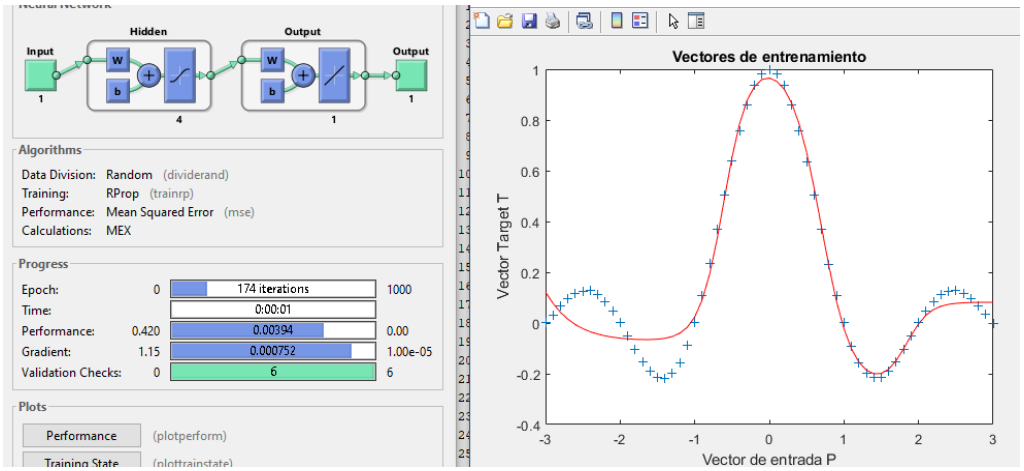


Fig. 1 Datos y gráfica del entrenamiento Trainrp con 4 neuronas.

- Trainlm(4 neuronas): Levenberg-Marquardt.

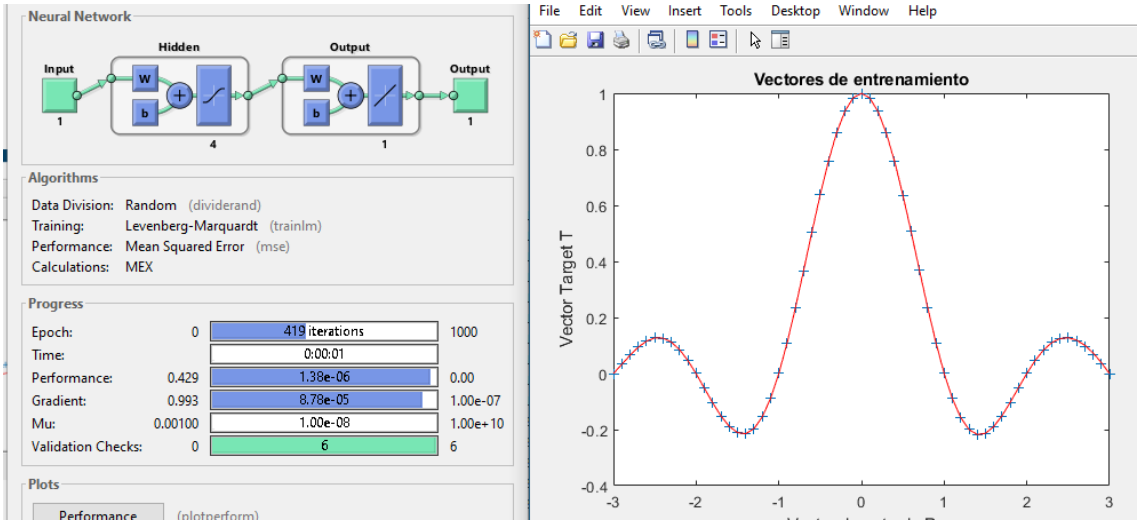


Fig. 2 Datos y gráfica del entrenamiento Trainlm con 4 neuronas.

- Trainbr (4 neuronas): Bayesian Regularization.

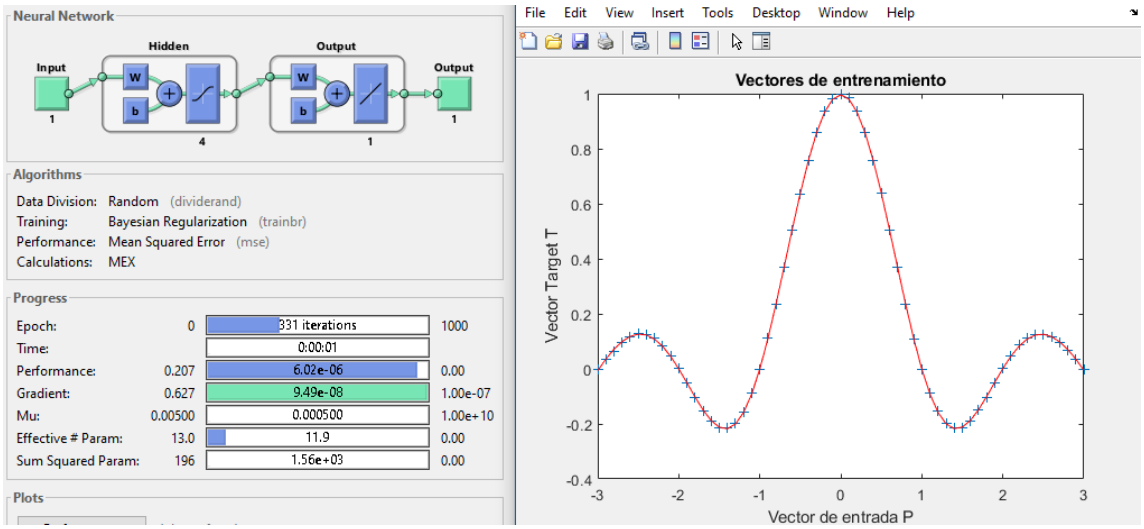


Fig. 3 Datos y gráfica del entrenamiento Trainbr con 4 neuronas.

- Trainbr (2 neuronas): Bayesian Regularization.

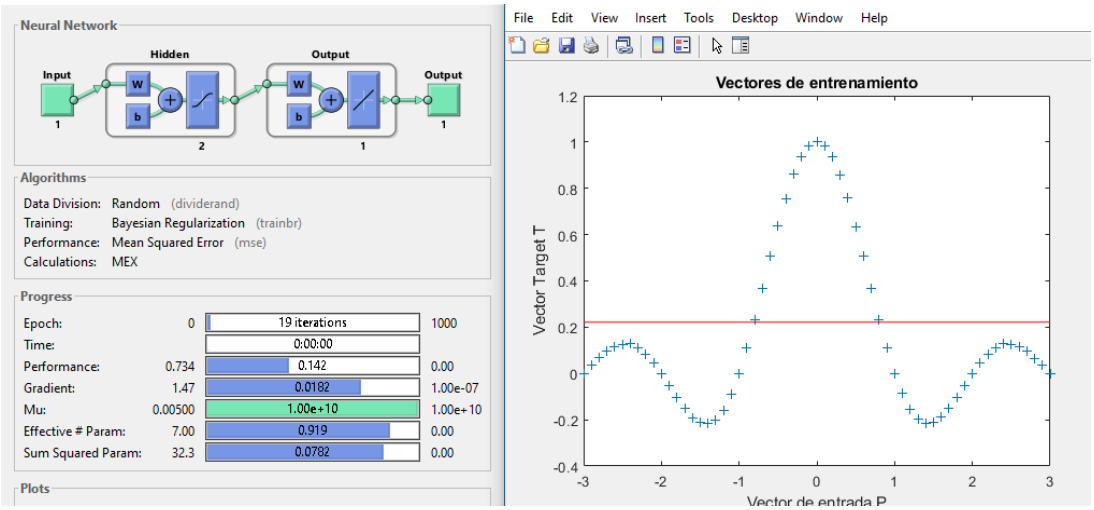


Fig. 4 Datos y gráfica del entrenamiento Trainbr con 2 neuronas.

- Traingd (4 neuronas): Gradient Descent.

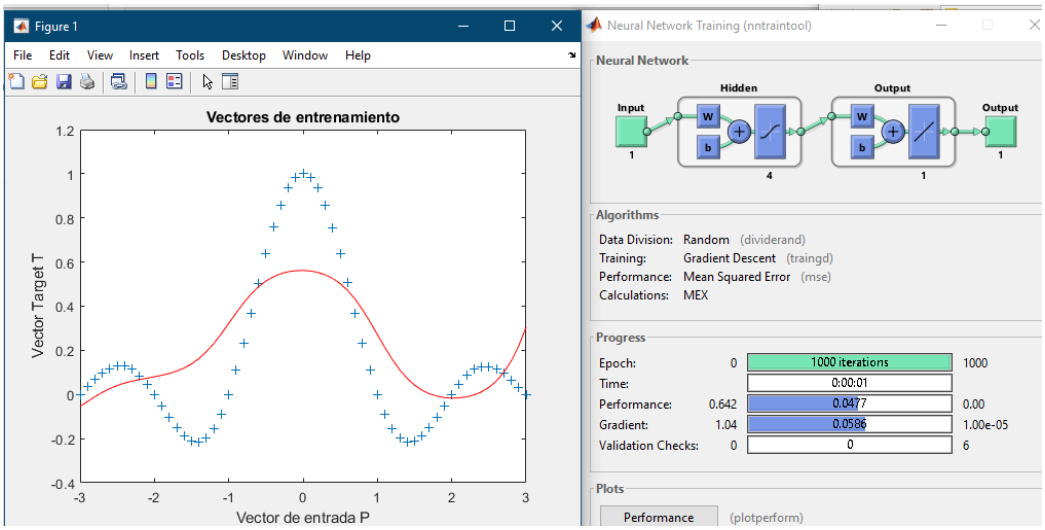


Fig. 5 Datos y gráfica del entrenamiento Traingd con 4 neuronas.

- Traingd (10 neuronas): Gradient Descent.

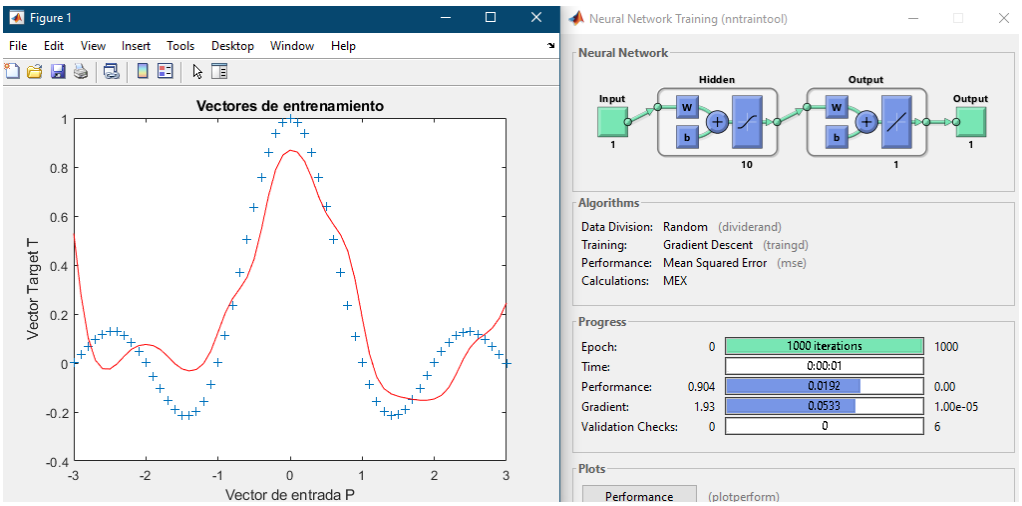


Fig. 6 Datos y gráfica del entrenamiento Traingd con 10 neuronas.

Conclusión

Según los resultados obtenidos, podemos decir lo siguiente.

- Los métodos de entrenamiento que más se acercan al resultado perfecto son: Trainbr *y* Trainlm. Tenemos que destacar que Trainlm, obtiene una mejor performance en el entrenamiento, ya que el valor obtenido en Trainlm es 1.30e-06 frente a 6.02e-06 del Trainbr, con respecto a las gráficas que sacan podemos ver como pasan por todos los puntos, y esto es gracias a que los dos se basan en la optimización Levenberg-Marquardt, el cual, es muy efectivo para entrenar redes con un resultado satisfactorio.
- El método que menos se aproxima al resultado es: Traingd, como podemos ver el algoritmo de descenso por la gradiente, no es muy efectivo para entrenar redes neuronales, ya que ofrece un resultado muy malo frente a otros tipos de algoritmos de entrenamiento.
- En el caso de poner 2 neuronas, vemos como no es capaz de acercarse al resultado, sin embargo, en el caso de 10 neuronas, vemos como mejora considerablemente el resultado, en este caso, lo hemos notado sobre entrenamiento, ya que mejora en todos los aspectos.

Ejercicio 3

En este ejercicio, vamos a comparar diversas graficas de diferentes métodos de entrenamiento, estas gráficas son: performance, estado del entrenamiento, histograma de los errores y la regresión. El entrenamiento con mejores resultados, le modificaremos la forma en la que se divide los datos, para así ver lo que sucede con las gráficas. Para ello vamos a explicar por encima, el objetivo de cada gráfica:

- Performance: Representa el error, test y validación en base al número de épocas de entrenamiento, por lo general, podemos ver cómo va bajando el error a medida que pasan las épocas, hasta que llega un punto en el que se mantiene. Esta gráfica siempre te redondeará el valor de la validación más bajo.
- Trainstate: Te muestra el estado del entrenamiento, mostrándote valores en base al número de épocas de entrenamiento, y también las comprobaciones de las validaciones y la época en la que se comprueba esas validaciones.
- Error Histogram: Te muestra un histograma, en el cual, podemos ver las instancias de los entrenamientos, validaciones, los test y la línea en el que el error es cero.
- Training Regression: En estas gráficas podemos ver la línea de regresión del entrenamiento, de las validaciones, de los test y de todo. Hay que mencionar que también te muestra el valor de la regresión.

Ahora vamos a proceder a realizar el estudio de los diferentes entramientos utilizados.

- Trainrp: Resilient backpropagation. En la gráfica de performance, para que tengamos un buen resultado, tiene que dar un resultado cercano a cero, porque el rendimiento se basa en el error cuadrático medio, en este caso nos da 15.7195 un valor alto. Si nos centramos en las gráficas de regresión se puede ver como tienen una buena regresión con una correlación de más de 0,8 en la mayoría de los casos. En cuanto, al histograma se ve como los errores no se encuentran cerca de la línea del 0 lo que indica que falla. Finalmente las gráficas de entrenamientos nos muestran la evolución del gradiente y las pruebas de validación hasta llegar 6 validaciones, que es la condición de parada.

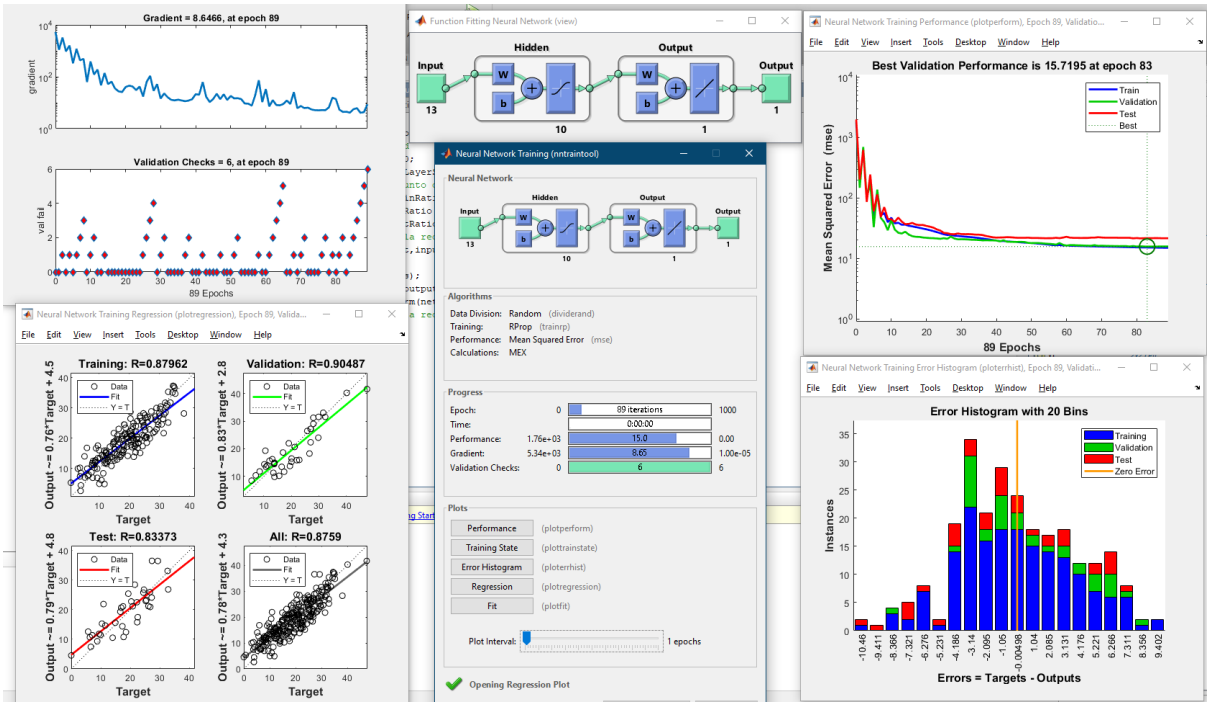


Fig. 1 Gráficas Trainrp.

- Trainlm: Levenberg-Marquardt. En la gráfica de la performance, para que tengamos un buen resultado, tiene que dar un resultado muy pequeño aproximado al 0, ya que este algoritmo de entrenamiento se basa en el algoritmo de mínimos de cuadrados. En este caso tiene un valor, 29,85. A diferencia del anterior los errores se encuentran esta vez más cercanos al 0 indicando un mayor acierto y el entrenamiento ha requerido de menos iteraciones. La correlación en este algoritmo sigue siendo bastante buena.

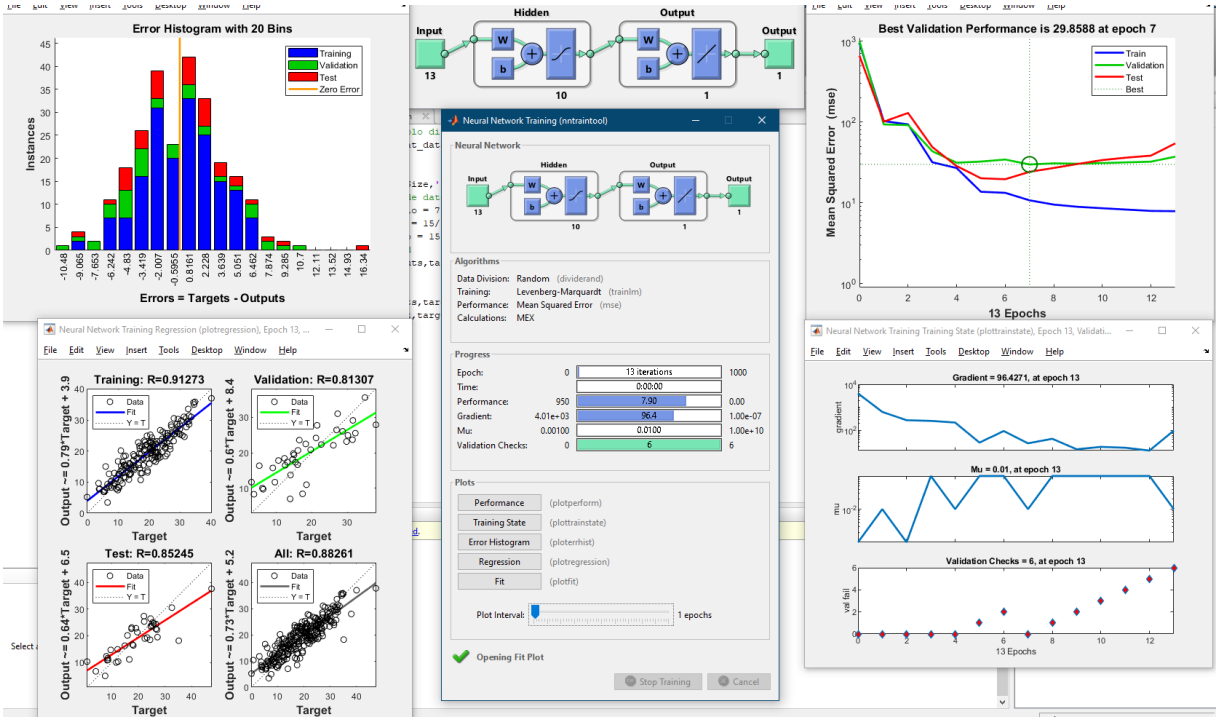


Fig. 2 Gráficas Trainlm.

- Trainbr: Bayesian Regularization. Le pasa lo mismo que a trainlm, cuanto menor sea el valor de la performance mejor, ya que se basa en el algoritmo de mínimos de cuadrados. En este caso tiene un valor de 14,18. Cabe destacar que al basarse en estadística bayesiana no requiere de validación alguna, lo que hace que tenga un entrenamiento diferente. Pero si nos centramos en los resultados vemos una aproximación al error cero un poco mala y una regresión buena de más de 0,8 de correlación.

Fig. 3 Gráficas Trainbr.

- Traingd: Gradient descent backpropagation. En este caso tiene un valor de 337, que es el inicial, además de verse que no es posible dibujar una recta de regresión al formar los puntos una bola. Por lo que será un descarte para este ejercicio.

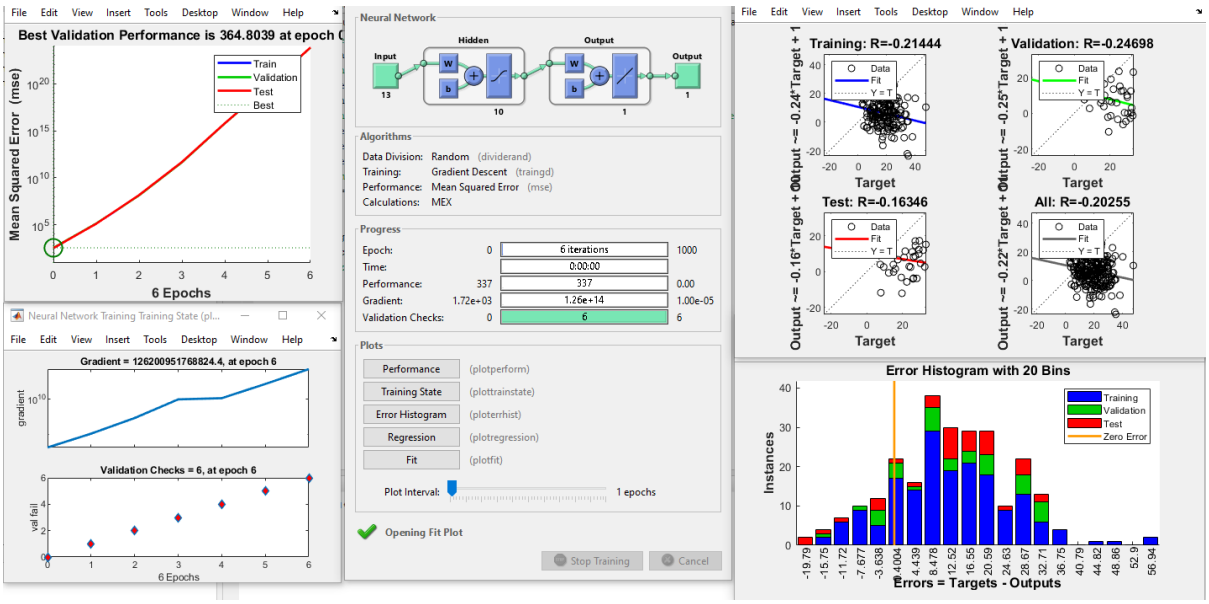


Fig. 4 Gráficas Traingd.

- Trainbfg: BFGS quasi-Newton backpropagation. En este caso el valor es de 14,6 con una recta de regresión que agrupa los datos en bastantes casos cuyo coeficiente de correlación es alto siendo mayor que 0,75.

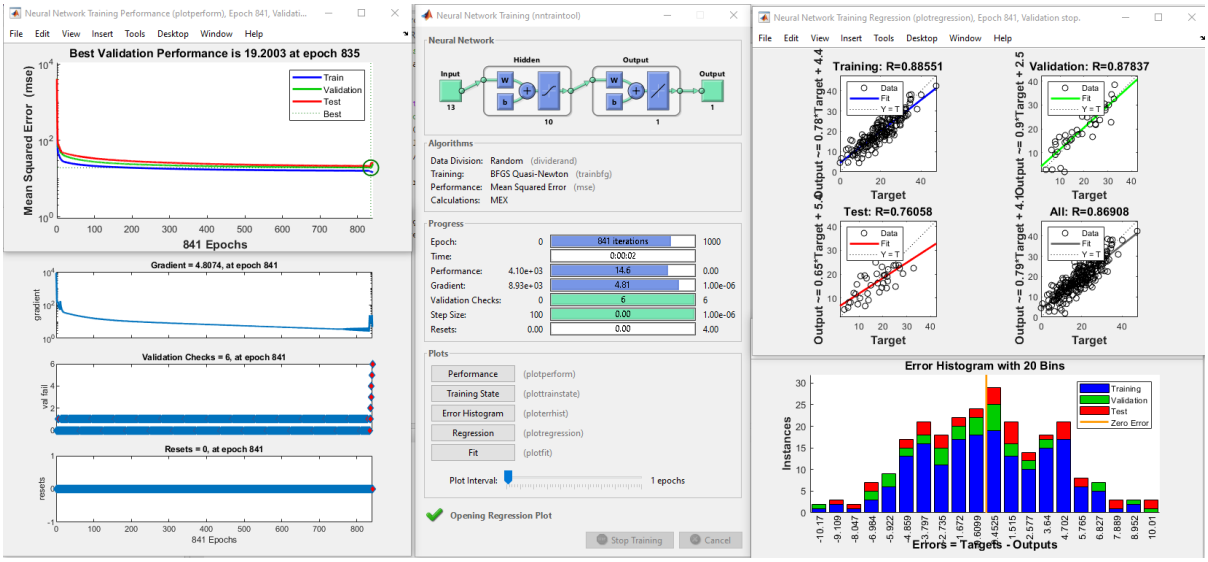


Fig. 5 Gráficas Trainbfg.

- Trainscg: Scaled conjugate gradient backpropagation. El valor de performance de este caso es de 18,6. La regresión es bastante buena, aunque el error no se acumula en el centro junto a la línea de error cero.

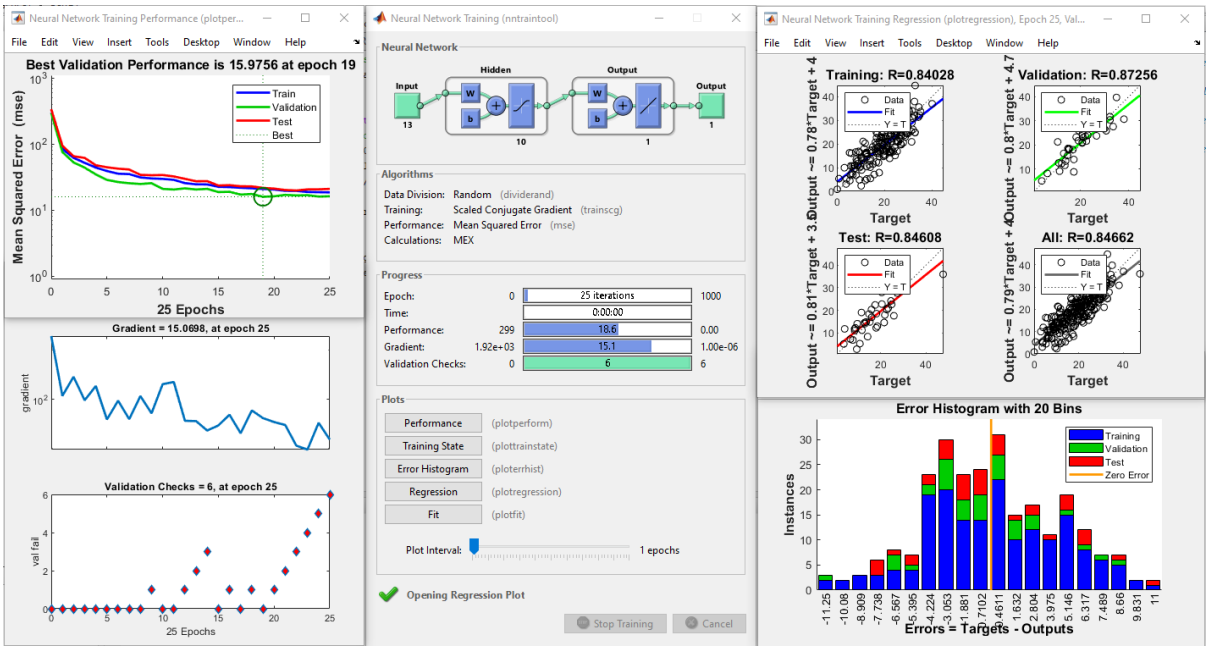


Fig. 6 Gráficas Trainscg.

En el caso de la gráfica fit, no se ha podido mostrar, ya que tiene varios elementos en los datos de entrada.

CONCLUSIÓN

Como hemos podido observar en los distintos entrenamientos estudiados en base a las gráficas mencionadas anteriormente. Vemos como cada algoritmo de entrenamiento obtienen resultados muy diferentes, esta conclusión vamos a dividirla por gráficas y diremos cuál es el mejor algoritmo de entrenamiento en cada apartado, y al final sacaremos cual es el mejor algoritmo de entrenamiento en general.

- Performance: El algoritmo que más se acerca al 0 es sin duda Trainbr, cuyo valor es 14.18, ligeramente superior al de Trainlm y Trainrp.
- Training State: Esta gráfica nos es relevante para ver como se ha comportado el entrenamiento, pero muchas veces no es realista, porque puede tomar un mínimo relativo como si fuera absoluto y acabar en pocas épocas, o al revés. Por tanto estos datos de épocas y validaciones son más de interpretación subjetiva.
- Error Histogram: El algoritmo con mejor gráfica es Trainlm, porque es el que más valores tiene próximos al error cero, estando agrupados cerca de su línea.
- Regresión: El algoritmo con la mejor regresión es Trainbr que empata junto a Trainlm aunque el resto de los algoritmos tienen una buena regresión al tener alta la correlación excepto Traingd.

Conclusión final, el algoritmo que mejor resultado nos ha dado en general, ha sido Trainlm. Como Trainlm es el mejor candidato, hemos decidido en modificar la división de datos de entrenamiento, para ver cómo afecta a su rendimiento, para ello hemos puesto los siguientes datos: trainRatio (50/100), valRatio(25/100), testRatio (25/100) y finalmente lo dividiremos con otra partición de datos, siendo la siguiente: trainRatio (60/100), valRatio (20/100), testRatio (20/100). Quedando el siguiente resultado:

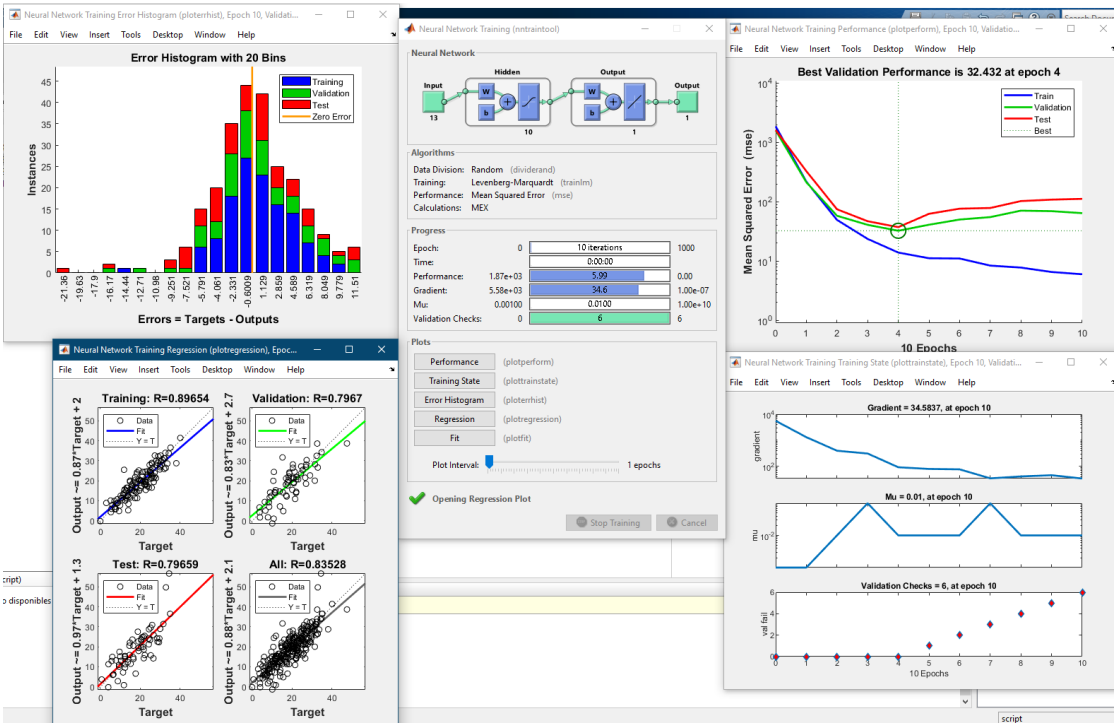


Fig. 7 Gráficas con división de datos modificado Trainlm.

Como podemos ver en los resultados, vemos que obtenemos mejores resultados en el histograma de errores y en el entrenamiento al ser más corto. Sin embargo la regresión es peor y el rendimiento también al tener un 0,84 de correlación total y un 32,43 de performance, respectivamente.

Ahora vamos a utilizar la siguiente división de datos: 60/20/20. Quedando los siguientes resultados:

Fig. 8 Gráficas con división de datos modificado Trainlm.

Aquí sucede lo mismo, obtenemos también mejores resultados en los errores y entrenamiento, pero la regresión y rendimiento son peores, con valores aún peores al anterior.
Se puede deducir de estas pruebas que si damos demasiados datos para la validación y test dejando pocos para entrenar es muy probable que estos tests y validaciones fallen, al tener una red poco entrenada, como se observa con las rectas y coeficientes de correlación de test y validación, lo que supone un peor rendimiento.

Ejercicio 4

Este ejercicio es muy similar al anterior, solo que vamos a analizar otro tipo de gráficas y datos sobre los mismos entrenamientos estudiados previamente, estas gráficas son: las matrices de confusión y curvas roc (características operativas del receptor). También, como en el anterior ejercicio, el entrenamiento con mejor resultado será el que le modificaremos la división de datos, para ver como repercuten en estas nuevas gráficas estudiadas.

- Trainrp: Levenberg-Marquardt.

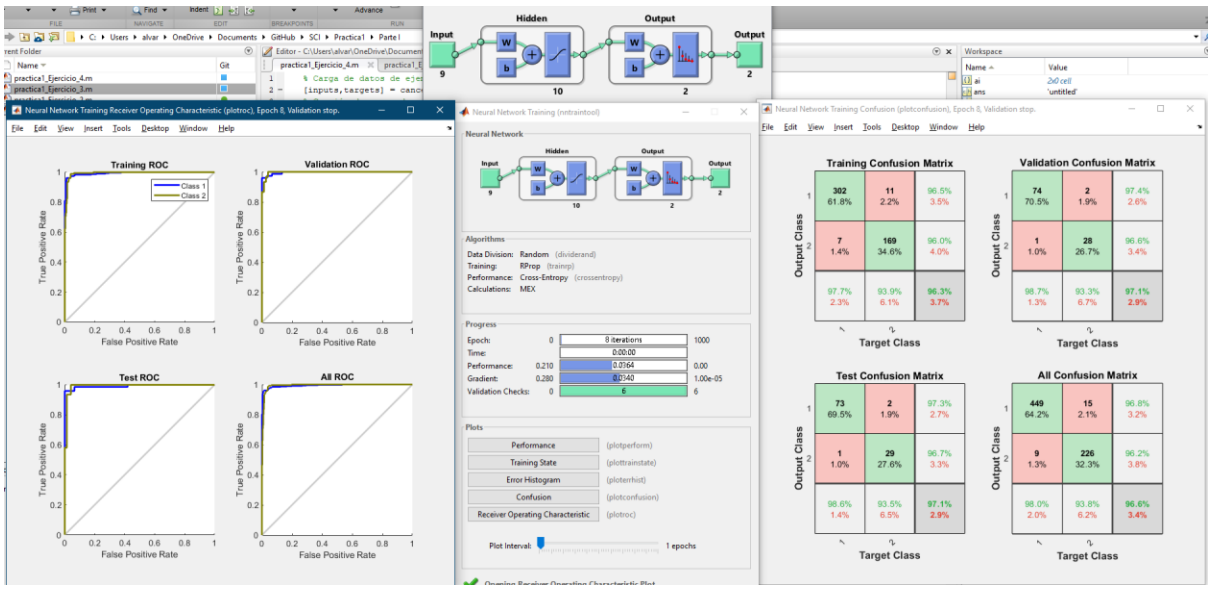


Fig. 1 Gráficas Trainrp.

- Trainlm: Levenberg-Marquardt.

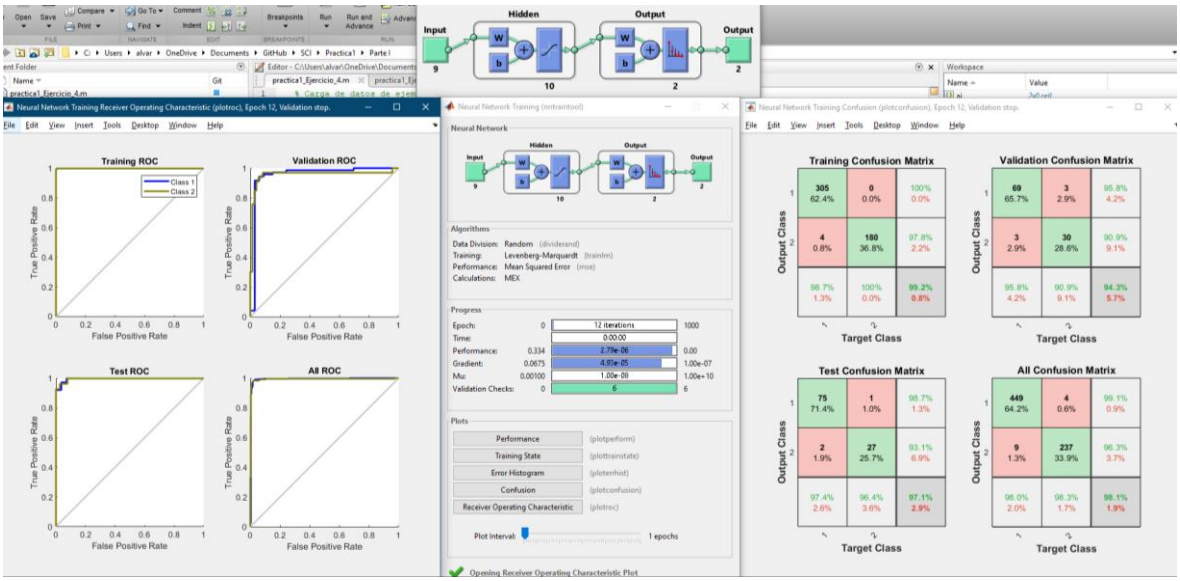


Fig. 2 Gráficas Trainlm.

- Trainbr: Bayesian Regularization.

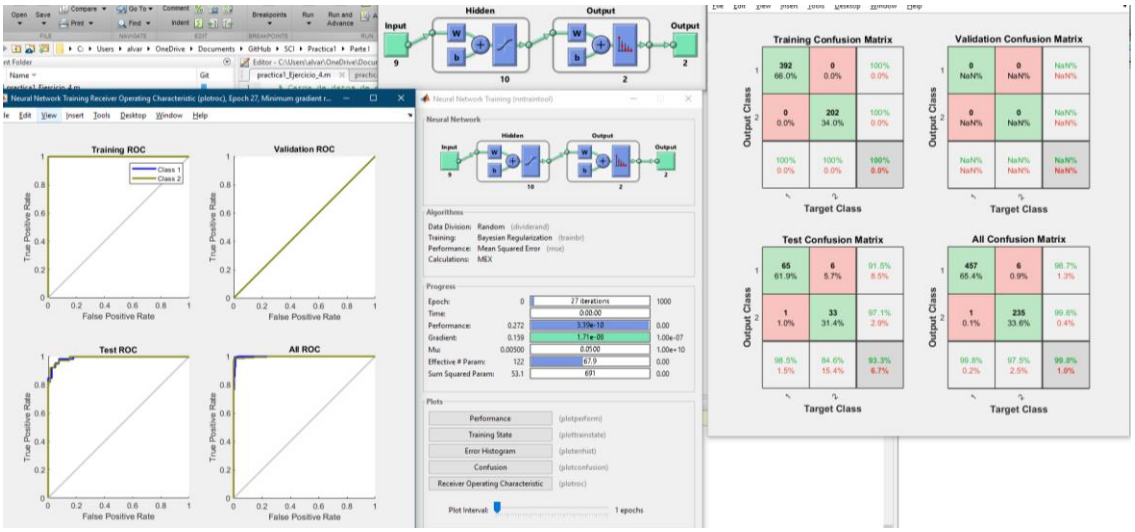


Fig. 3 Gráficas Trainbr.

- Traingd: Gradient descent backpropagation.

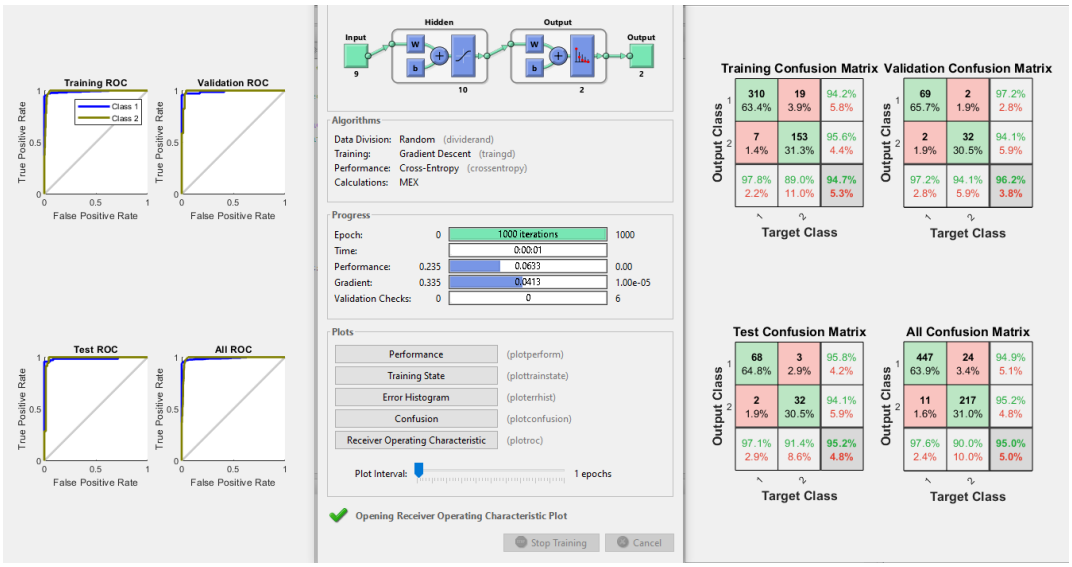


Fig. 4 Gráficas Traingd.

- Trainbfg: BFGS quasi-Newton backpropagation.

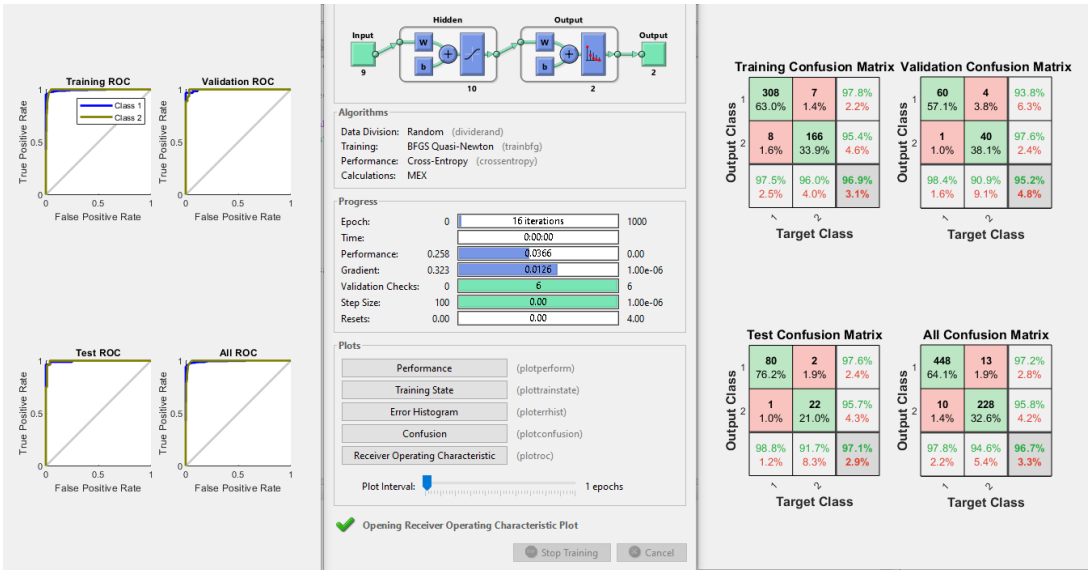


Fig. 5 Gráficas Trainbfg.

- Trainscg: Scaled conjugate gradient backpropagation.

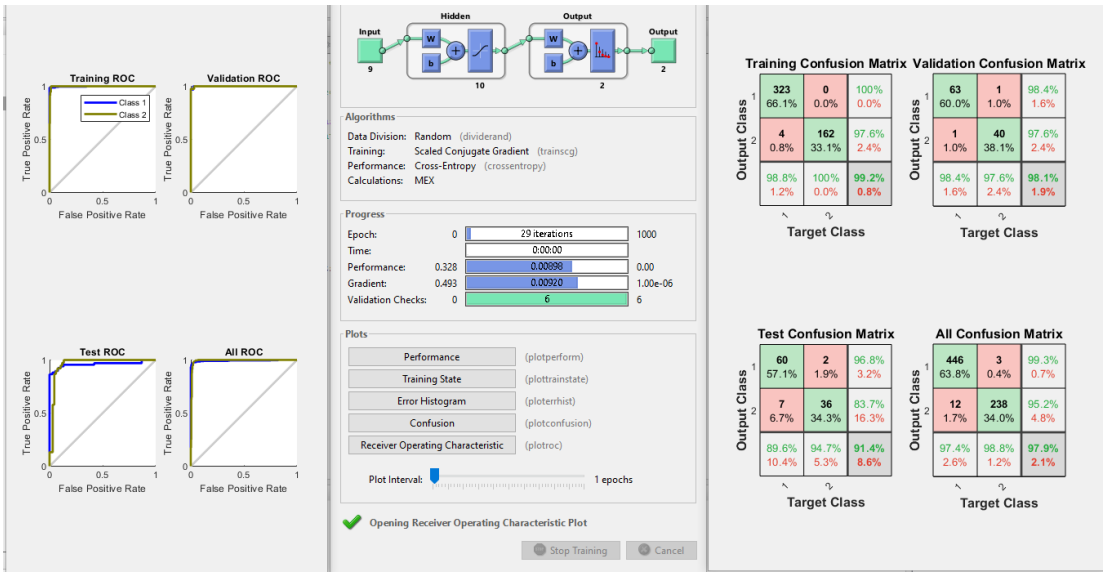


Fig. 6 Gráficas Trainscg.

Cabe destacar, que las gráficas ROC, nos dan en todos los algoritmos de entrenamiento, resultados muy buenos, casi siempre por encima del 0.9, ya que esto se puede saber si nos fijamos en la línea diagonal que separa la gráfica, si nuestras líneas de clases están por arriba de esta línea diagonal, los resultados son buenos, y cuanto más próximo al 1 serán mucho mejores. También hay que resaltar que en las matrices de confusión, también obtenemos resultados muy buenos, ya que el éxito siempre tiene más de un 90%.

CONCLUSIÓN

Como podemos ver en este estudio de gráficas, hemos obtenido muy buenos resultados, siendo muy difícil seleccionar el que más acierto tiene, aunque destaca ligeramente el Trainbr con un 99% de acierto en todas las matrices de confusión, teniendo por detrás a Trainlm con un 98%.

Por lo que se hará sobre el algoritmo de entrenamiento: Trainbr. También se hará con la división de datos: 80/20 y 70/30, debido a que no requiere validación al basarse en estadística bayesiana.

En la división 80/20 hemos obtenidos los siguientes resultados:

Fig. 7 Gráficas con división de datos modificado Trainbr.

Como podemos apreciar, los resultados son bastante buenos, aunque comparando, en las gráficas ROC, se obtiene mejores resultados, aunque siguen siendo muy buenos, con respecto a las matrices, también se obtienen mejores resultados.
En la división 70/30, hemos obtenido los siguientes resultados:

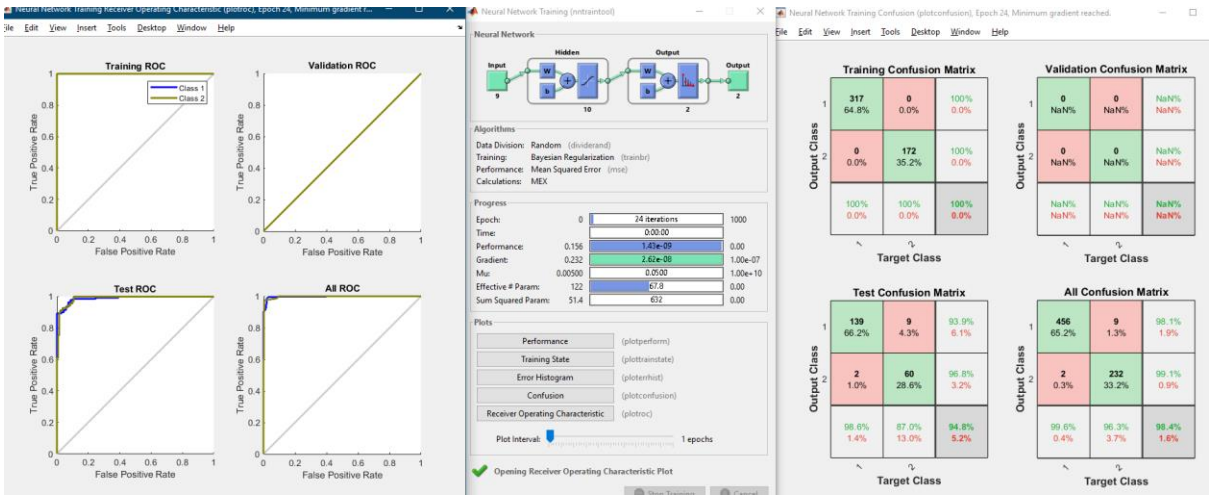


Fig. 7 Gráficas con división de datos modificado Trainbr.

Como podemos comprobar en los resultados obtenidos, los resultados siguen siendo muy buenos, aunque en las gráficas ROC ha empeorado los resultados, con respecto a la anterior división de datos, y está muy igualada con respecto a la división original de datos, luego, con respecto a las matrices, podemos apreciar que están muy equilibradas con respecto a la división original de datos. Al igual que en el ejercicio anterior al destinar más datos al entrenamiento podemos ver como mejora la puntuación.

Parte II

Ejercicio 1

En esta parte el objetivo es, diseñar un sistema que simule el desplazamiento del robot hasta unas coordenadas mediante un controlador, que trataremos como una caja negra. Lo primero ha sido crear un diagrama de bloques, que es el siguiente:

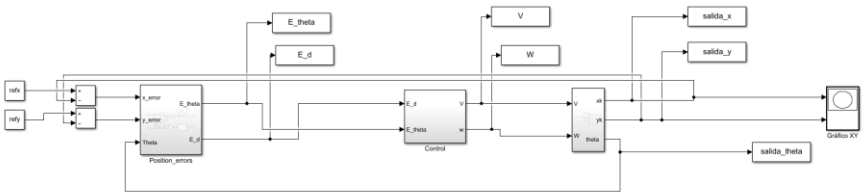


Fig. 1 Diagrama del sistema.

Luego hemos programado Position_errors en base a las condiciones y fórmulas que nos decía el enunciado, quedando como se muestra en la siguiente imagen, añadiendo un módulo de parada en función al error y otro para limitar ángulos a [-pi,pi]:

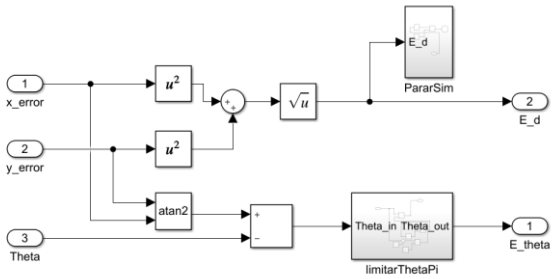


Fig. 2 Diagrama de Position_errors.

Una vez configurado y programado todo, el robot es capaz de moverse y hacer una trayectoria, como se puede apreciar en la siguiente imagen:

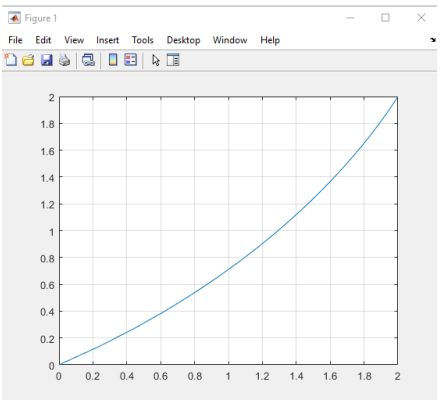


Fig. 3 Gráfica trayectoria.

En este caso, hemos programado al robot para que vaya desde la posición x = 0 e y = 0 a la posición x = 2 e y = 2. Por lo tanto, podemos decir, de que el robot está bien programado, ya que llega al punto, el cual, le hemos dicho que vaya.

Ejercicio 2

En este ejercicio, tenemos que crear una red neuronal que emule la caja negra del ejercicio anterior, actuando como un neurocontrolador.

a) En el primer apartado, tenemos que configurar el diagrama, limitando la duración a 100 segundos de la simulación, añadiendo la variable Ts como periodo de muestreo y poniendo el solver en ode3 y Fixed-step.

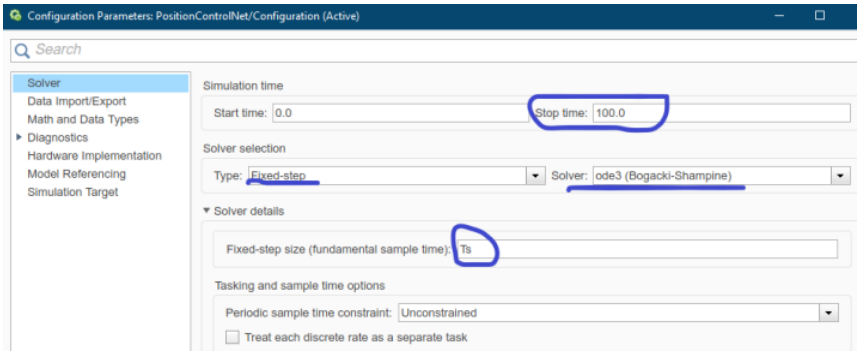


Fig. 1 Configuración del diagrama del sistema.

b) Seguidamente, copiamos el código proporcionado por el profesor, que simulará el diagrama desde el entorno de comandos de Matlab.

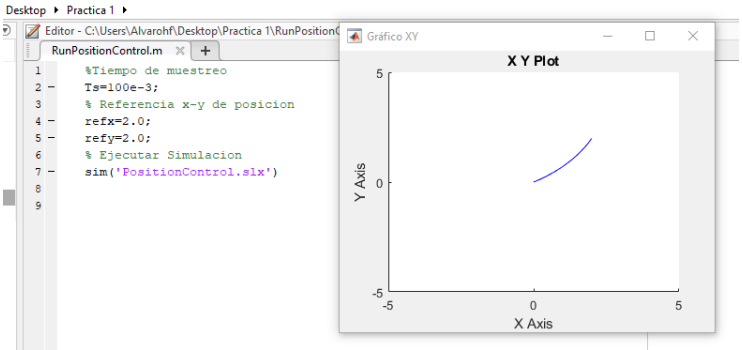


Fig. 2 Código y trayectoria del robot.

c) Ahora vamos a comprobar que se generan correctamente las variables que contienen las salidas y entradas del controlador. Viendo que se han creado correctamente y contienen datos y tiempos.

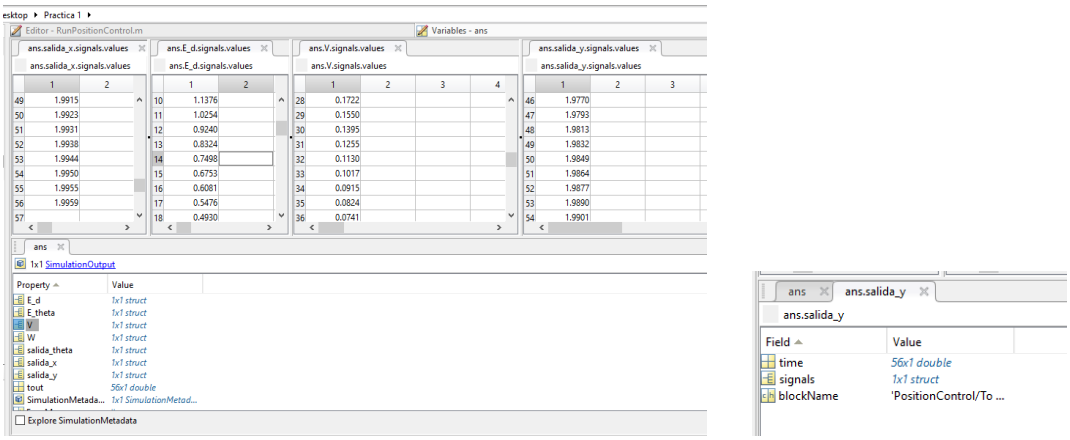


Fig. 3 Variables entradas y salidas.

d) Ahora se ejecuta y vemos la trayectoria que hace el robot, con una gráfica creada a partir de los puntos generados en la simulación.

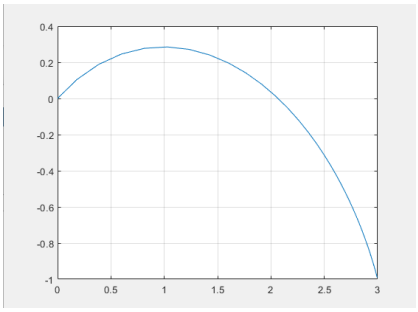


Fig. 4 Trayectoria robot.

e) Ahora repetiremos los procesos anteriores, pero con diferentes datos y muchas más veces para así generar un conjunto de datos que servirá para entrenar a la red.

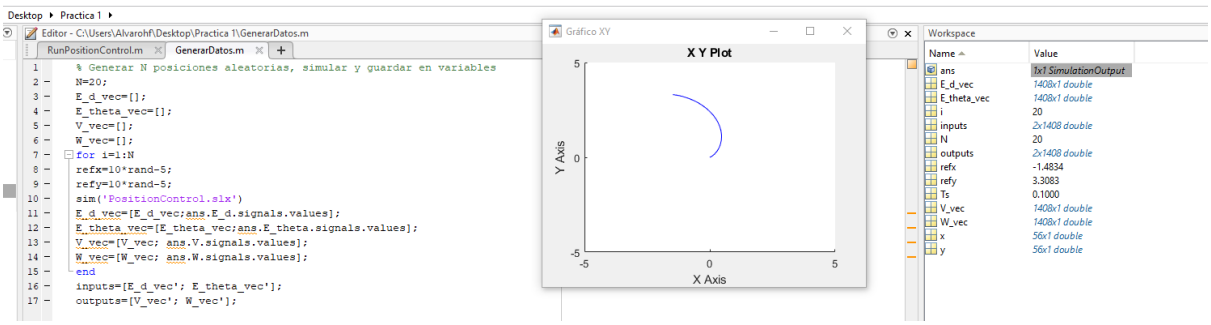


Fig. 5 Código, trayectoria del robot y los datos generados.

f) Después se nos pide que entrenemos una red para sustituir el módulo de control del sistema del robot, para ello usando los métodos vistos anteriormente, obteniendo una red que es capaz de llegar al punto de destino a partir del origen.

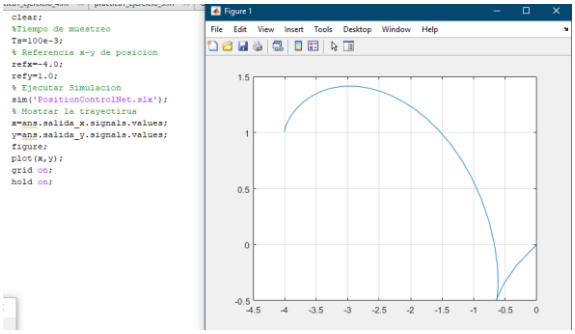


Fig. 6 Trayectoria del robot con la red neuronal creada.

g) Como último apartado de este ejercicio, tenemos que comparar el resultado de la red neuronal, con la caja negra para diferentes puntos. Para ello, hemos creado un script, el cual realiza 5 comparaciones (se pueden poner las que se quieran):

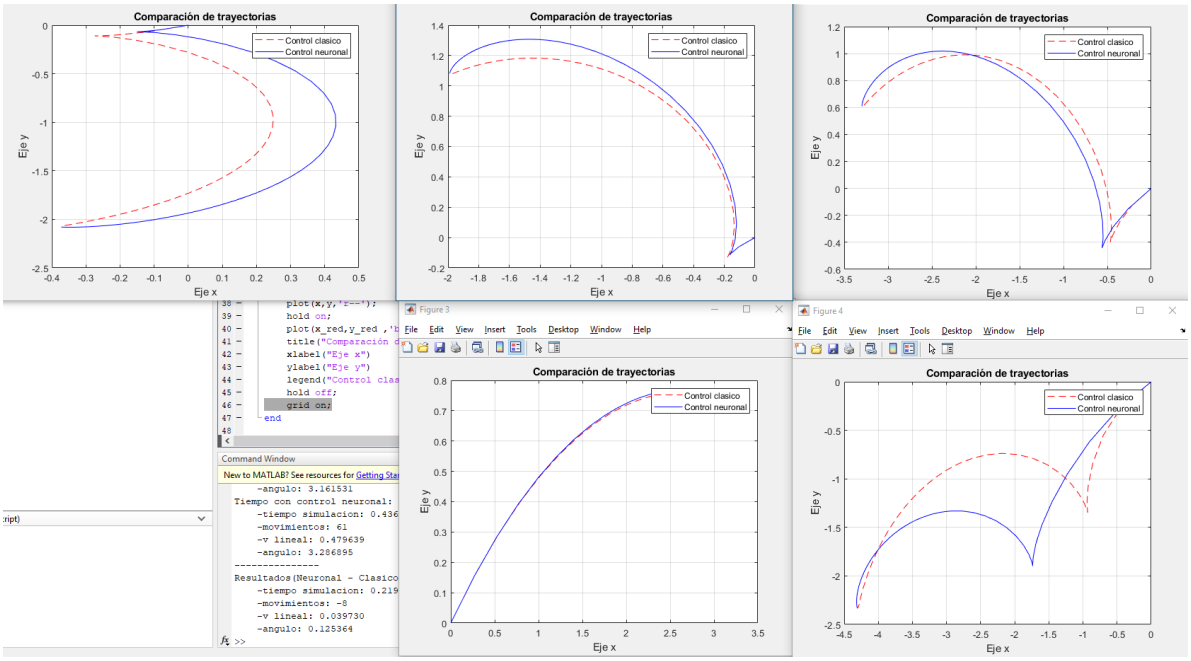


Fig. 7 Comparación de diferentes trayectorias con el control clásico y el control neuronal.

Podemos apreciar, que el módulo de control neuronal funciona correctamente llegando siempre al punto de destino, aunque en muchos casos difiere de la ruta seguida por el módulo de control original. Para poder realizar más comparaciones hemos medido también tiempos, velocidades lineales, movimientos y ángulos.

```
Comparacion: 1
====================
Control clasico:
    -tiempo simulacion: 0.202962s
    -movimientos: 64
    -v lineal: 0.378503
    -angulo: 2.929756
Tiempo con control neuronal:
    -tiempo simulacion: 0.892122s
    -movimientos: 58
    -v lineal: 0.447034
    -angulo: 3.167590
---------------
Resultados(Neuronal - Clasico):
    -tiempo simulacion: 0.689160s
    -movimientos: -6
    -v lineal: 0.068530
    -angulo: 0.237833
Comparacion: 2
====================
Control clasico:
    -tiempo simulacion: 0.288312s
    -movimientos: 64
    -v lineal: 0.328377
    -angulo: -1.969485
Tiempo con control neuronal:
    -tiempo simulacion: 0.190063s
    -movimientos: 61
    -v lineal: 0.409087
    -angulo: -2.126594
---------------
Resultados(Neuronal - Clasico):
    -tiempo simulacion: -0.098248s
    -movimientos: -3
    -v lineal: 0.080710
    -angulo: -0.157109
```

```
Comparacion: 3
====================
Control clasico:
    -tiempo simulacion: 0.201497s
    -movimientos: 57
    -v lineal: 0.607178
    -angulo: -0.033436
Tiempo con control neuronal:
    -tiempo simulacion: 0.179174s
    -movimientos: 48
    -v lineal: 0.722488
    -angulo: -0.024334
---------------
Resultados(Neuronal - Clasico):
    -tiempo simulacion: -0.022323s
    -movimientos: -9
    -v lineal: 0.115310
    -angulo: 0.009102
Comparacion: 4
====================
Control clasico:
    -tiempo simulacion: 0.208786s
    -movimientos: 72
    -v lineal: 0.352016
    -angulo: 3.696296
Tiempo con control neuronal:
    -tiempo simulacion: 0.181464s
    -movimientos: 65
    -v lineal: 0.095835
    -angulo: 3.630026
---------------
Resultados(Neuronal - Clasico):
    -tiempo simulacion: -0.027322s
    -movimientos: -7
    -v lineal: -0.256180
    -angulo: -0.066270
```

```
Comparacion: 5
====================
Control clasico:
    -tiempo simulacion: 0.216364s
    -movimientos: 69
    -v lineal: 0.439909
    -angulo: 3.161531
Tiempo con control neuronal:
    -tiempo simulacion: 0.436104s
    -movimientos: 61
    -v lineal: 0.479639
    -angulo: 3.286895
---------------
Resultados(Neuronal - Clasico):
    -tiempo simulacion: 0.219740s
    -movimientos: -8
    -v lineal: 0.039730
    -angulo: 0.125364
```

Fig. 8 Comparación datos control clásico y control neuronal.

En estos resultados vemos que, en el tiempo, es mucho menor en el caso del control neuronal al igual que el número de movimientos, esto se debe a que tiende

a hacer menos giros y más avances en línea recta, como las medias de ángulos corroboran. Teniendo en cuenta esto vemos que la velocidad lineal es mayor en el caso de la red neuronal, motivo que explica porque la simulación la realiza en un menor tiempo que la del control normal. Estos datos tienen que verse con cuidado porque se tratan de medias, que pueden no representar correctamente al conjunto de datos.

Parte III

Ejercicio 1

En este ejercicio nos piden que identifiquemos un sistema utilizando una red recursiva de tipo NARX. Para ello la primera parte consiste en crear un sistema que emule la respuesta del sistema, se compone de un generador de datos, una caja negra y un Scope para visualizar en forma de ondas los datos. Lo podemos ver en la siguiente imagen.

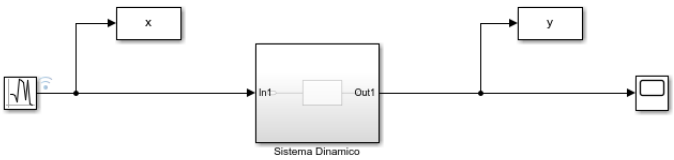


Fig. 1 Diagrama del sistema.

Con el sistema creado, lo configuramos poniendo un tiempo de simulación de 30s la frecuencia como la variable Ts y el solver con Fixed-step (Para poder poner Ts), además de tiempo discreto.

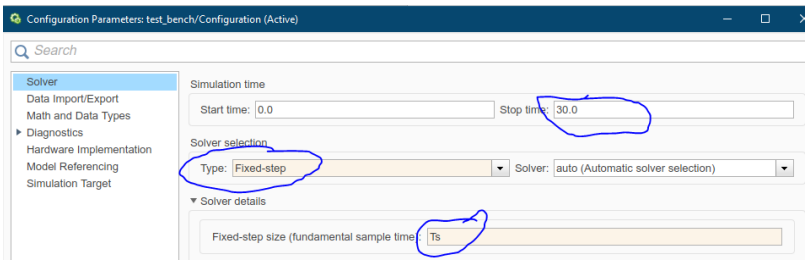


Fig. 2 Configuración del sistema creado.

Una vez configurado, copiamos el código que nos dan en la práctica para así poder entrenar una red NARX. La red entrenada obtuvo los siguientes datos, que son buenos resultados al tener una buena perfomance muy próxima al 0:

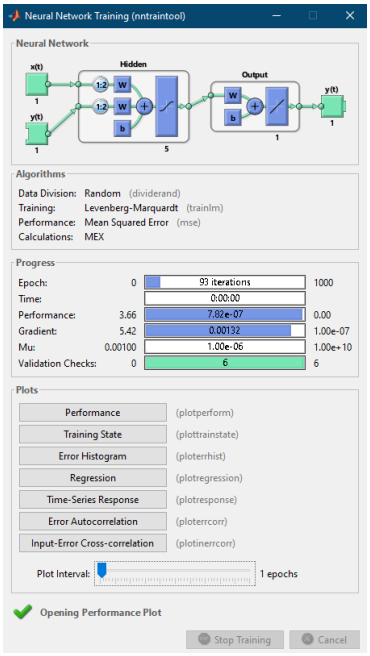


Fig. 3 Resultados del entrenamiento de la red NARX.

Una vez obtenida la red NARX entrenada, modificamos el sistema anterior mostrado, añadiendo la NARX entre el Scope y la entrada x para así comparar ambos sistemas.

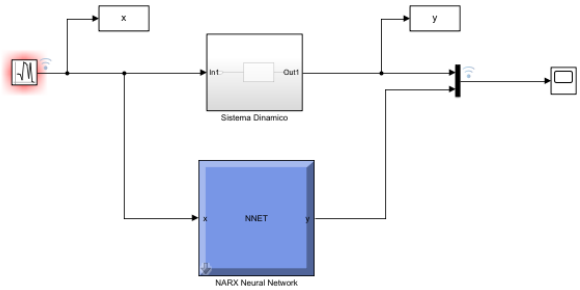


Fig. 4 Diagrama de bloques del sistema con la red NARX.

Dado este sistema, podemos ver como la red generada y el modelo de sistema, dan casi los mismos resultados en la gráfica.

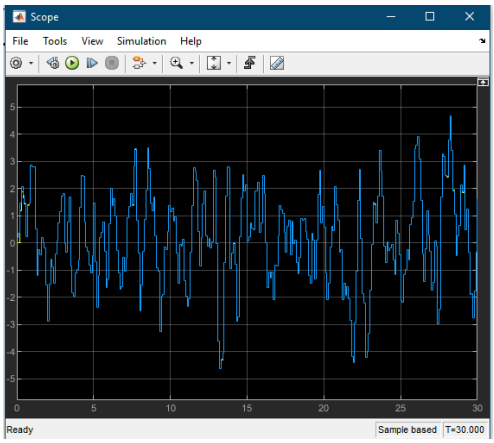


Fig. 4 Gráfica de la rede generada y el modelo del sistema.

Como se puede apreciar, el error es muy pequeño de la red NARX frente al modelo original, cabe destacar que al principio es donde más error hay, ya que es donde menos coinciden las líneas del modelo original y la red NARX entrenada, en el resto de los casos coincide a la perfección, viendo así que la red NARX está bien entrenada y funciona correctamente en este caso.

Ejercicio 2

a) Para la creación del esquema de simulink hemos utilizado el modelo anterior de la parte control, sustituyendo el módulo de control por el nuevo, y el sistema de referencia con el TrajectoryGeneration.slx. Además, en Matlab declaramos en un script los valores iniciales para x_0, y_0 y theta_0.

b) En esta parte hemos creado el script RunTrajectoryControl.m en el que aparte de indicar los valores iniciales, hemos sacado las posiciones actuales y las de referencia (exportándolas del simulink al workspace), para compararlas en un gráfico usando plot y hold on para mostrar ambas. Al hacer esta comparación podremos ver si se asemeja al objetivo que en este caso serían los puntos de la trayectoria generada, que como podemos ver es bastante precisa, excepto en algunas zonas donde difiere un poco al comienzo.

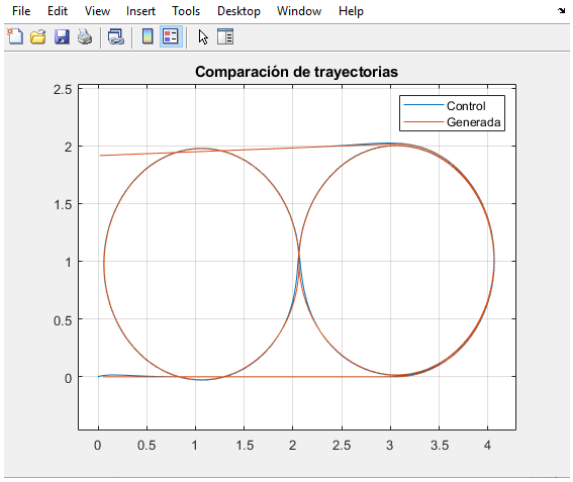


Fig. 1 Comparación de las trayectorias del sistema.

c) Siguiendo como base el ejemplo del ejercicio anterior hemos entrenado varios modelos neuronales, que han requerido de bastantes pruebas de acierto y error, con distintas neuronas. Al tener que sustituir solo el controlador los datos que nos interesaban eran los errores como entradas y las velocidades como salidas. Otro cambio que hemos tenido que hacer ha sido el tamaño de los arrays de tipo cell, haciendo que tuvieran tamaño 2 para permitir que introducir los dos errores y las dos velocidades dejándolo así mat2cell (inputs, 2, ones(nT,1)).
Realizando bastantes entrenamientos obtuvimos bastantes redes, muchas de ellas no funcionaban correctamente, y muy pocas que lo hacían relativamente bien hasta que se cambiaban las posiciones. Por lo que optamos por generar más datos con distintas referencias para solventar el problema, obteniendo así un modelo con un buen rendimiento y cuyo funcionamiento en la práctica era bueno y permitía variar

sus posiciones iniciales, aunque solo funciono bien en la de 10 neuronas.
Los criterios de selección de las redes por experimentación se basaban en una primera etapa en el performance de la red, que era el error cuadrático medio, escogiendo aquellos que tuvieran uno muy cercano al cero, pero también era necesario comprobar que la validación se realizará correctamente y no nos produjera una red sobreentrenada, que memorizase los datos de entrenamiento. Esto último lo hicimos viendo las gráficas que plotperform ofrecía y viendo cuando se paraba y como estaban los datos en comparación al resto. Con estas redes entrenadas generamos un bloque de simulink que pusimos en la red para probar finalmente, escogiendo así los mejores teóricamente. Las neuronas que escogimos para los distintos modelos fueron de 5, 10, 15 y 20 neuronas, trabajando con un único dato generado y después con un conjunto de 50 datos. Como se observa la mayoría con un único dato ofrecen una performance muy buena al ser cercana al 0 destacando la de 15 neuronas con un error cuadrático de 2.71e-06.

Fig. 2 Entrenamiento redes de 5 y 10 neuronas de un solo dato.

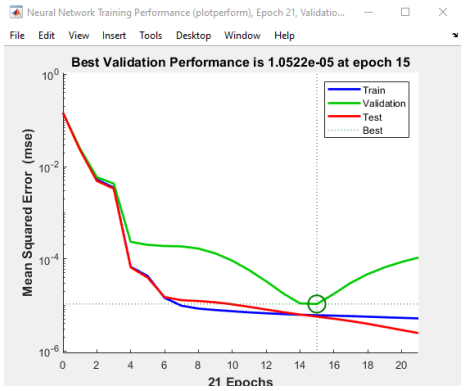


Fig. 3 Validación de la red de 5 neuronas de un solo dato.

Fig. 4 Entrenamiento redes de 15 y 20 neuronas de un solo dato.

Con las de múltiples datos este error ya sube bastante porque aumenta la dificultad del entrenamiento de la red. El mejor resultado de rendimiento se obtiene con la de 20 neuronas con un 0.008 y el peor con la de 5 neuronas con un 0.0123, siendo la diferencia muy alta con respecto a las demás.

Fig. 5 Entrenamiento redes de 5 y 20 neuronas de múltiples datos.

d) En esta parte lo que hemos hecho ha sido un pequeño script para comparar superponiendo las trayectorias del control original sobre el control de la red neuronal, para así observar si las redes en la práctica son igual de efectivas. Para ello hemos cogido los modelos anteriores para hacer prueba y error y ver los resultados. Sin embargo, la mayoría de los casos a pesar de tener un rendimiento teórico alto con una buena validación distan de aproximarse a los resultados reales salvo el caso de la red de 10 neuronas de múltiples datos, el resto o realizan

trayectorias aleatorias o imitan la que deben recorrer erróneamente. En definitiva se trata de realizar muchas pruebas hasta tener suerte y dar con una buena red, aunque con ciertas modificaciones se puede dar con resultados mejores (más neuronas, más datos, etc.).

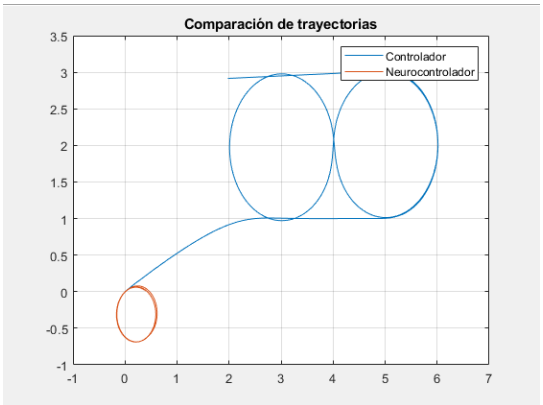


Fig.6 Trayectoria aleatoria generada por la red de 5 neuronas y un solo dato.

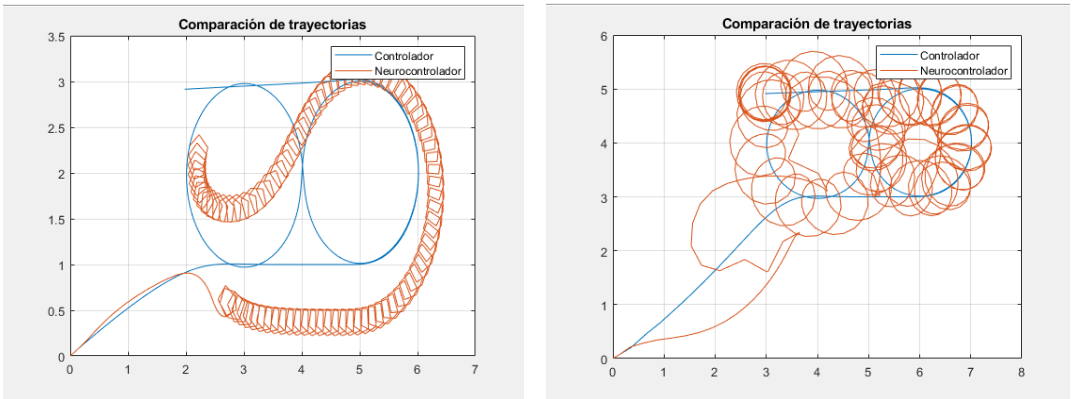
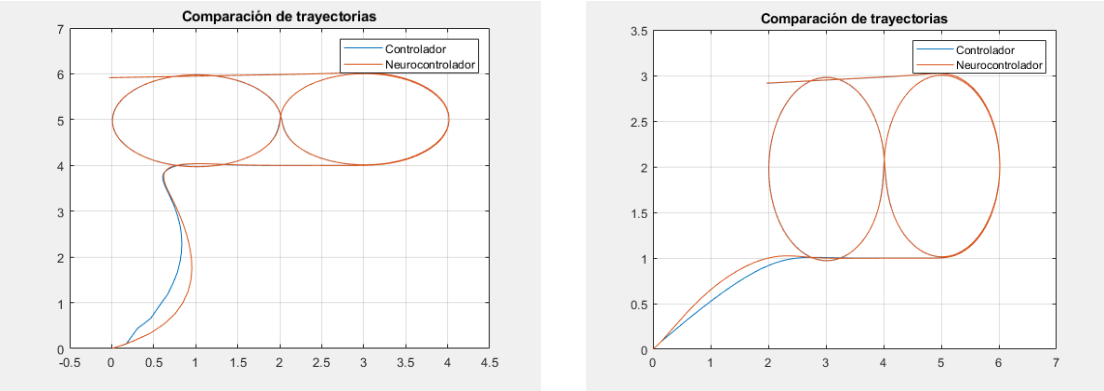


Fig. 1 Trayectorias erróneas generadas por las redes de 5 y 15 neuronas de múltiples datos.



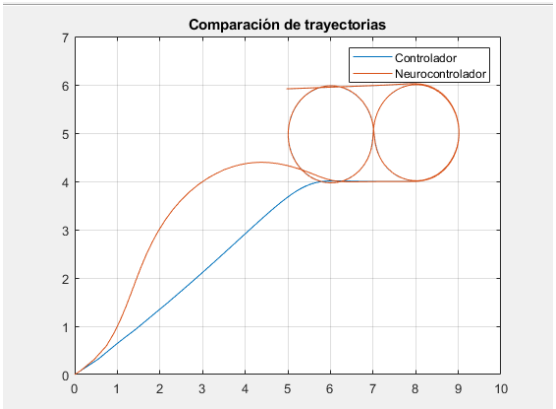


Fig. 7 Trayectorias aproximadas generadas por la red de 10 neuronas y múltiples datos.

3.Conclusión

Se puede concluir viendo todas las resoluciones de los ejercicios que es vital elegir un buen método de entrenamiento para obtener una red neuronal adecuada al problema. Además de realizar una división acorde a los datos, para entrenar correctamente los datos junto al número adecuado de neuronas. Unas de las redes más útiles son las NARX que son de lazo cerrado y se emplean bastante en control como se ha podido ver en el ejemplo, principalmente gracias a su retroalimentación, colocándose entre el controlador que se desea replicar.

Aunque muchas veces se requerirá de experimentación para encontrar la red neuronal adecuada, porque no siempre encuentra el mínimo absoluto o se sobre entrena entre muchos otros motivos.

4.Bibliografía

- https://es.mathworks.com/help/map/ref/wraptopi.html
- https://es.mathworks.com/help/deeplearning/ref/trainrp.html
- https://es.mathworks.com/help/deeplearning/ref/traingd.html
- https://es.mathworks.com/help/deeplearning/ref/trainscg.html
- https://es.mathworks.com/help/deeplearning/ref/trainbfg.html
- https://es.mathworks.com/help/deeplearning/ref/trainbr.html
- https://es.mathworks.com/help/deeplearning/ref/trainlm.html