eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

ESTUDIO Y DESARROLLO DE MODELOS NEURONALES DE COMPLEJOS SISTEMAS MULTIVARIABLES

Estudiante: ALONSO MEDINA, AIMAR
Director/a: IRIGOYEN GORDO, ELOY
Departamento: Ingeniería de Sistemas y Automática
Curso académico: 2021-2022

Bilbao, 12 de septiembre de 2022

Resumen Laburpena Abstract

El objetivo del presente proyecto es el estudio y desarrollo de modelos neuronales de complejos sistemas multivariables. Tras el modelado se obtendrán los modelos neuronales de dos sistemas no-lineales objetivo, uno monovariable, y otro multivariable. Además, se obtendrá un código que permita optimizar el desarrollo de más modelos, para su aplicación en estrategias de control dependiente de modelos dinámicos. Para esto, se han escogido redes neuronales de estructura NARX, y el desarrollo se ha realizado en el entorno de MATLAB.

Palabras Clave: identificación, red neuronal, sistemas no-lineales, modelo NARX, MIMO.

Proiektu honen helburua aldagai anitzeko sistema konplexuen eredu neuronalak aztertzea eta garatzea da. Modelaketaren ostean, bi sistema ez-lineal lortuko dira, bata aldagai bakarrekoa, eta bestea, aldagai anitzekoa. Horrez gain, eredu gehiagoren garapena optimizatzea ahalbidetuko duen kodea lortuko da, eredu dinamikoetan oinarritzen diren kontrol-estrategietan erabiltzeko. Honetarako, NARX estruktura duten sare neuronalak aukeratu dira, eta garapena MATLAB ingurunean egin da.

Gako-hitzak: identifikazioa, sare neuronala, sistema ez-linealak, NARX eredua, MIMO.

The objective of this project is the study and development of neural models of complex multivariable systems. After the modelling, the neural models of two target nonlinear systems will be obtained, one SISO model, and one MIMO model. Moreover, a code for streamlining future development of further models, for their application in dynamic model based control strategies. For this, neural networks with the NARX structure have been chosen, and the development has been conducted on the MATLAB enviroment.

Keywords: identification, neural network, nonlinear systems, NARX model, MIMO.

Índice

Lista de figuras

Lista de tablas

Lista de acrónimos

ARX Autoregressive eXogenous (Autorregresivo eXógeno)

FIR Finite Impulse Response (Respuesta Finita al Impulso)

MaAE Maximum Absolute Error (Error Absoluto Máximo)

MIMO Multiple Input Multiple Output (Múltiples Entradas Múltiples Salidas)

MLP Multi-Layer Perceptron (Perceptrón MultiCapa)

MPC Model Predictive Control (Control Predictivo por Modelo)

MSE Mean Square Error (Error Cuadrático Medio)

NARX Nonlinear Autoregressive eXogenous (No-lineal Autorregresivo eXógeno)

OE Output Error (Error de Salida)

RNA Redes Neuronales Artificiales

SISO Single Input Single Output (Una Entrada Una Salida)

1. Introducción

Este documento es la memoria o resultado del Trabajo de Fin de Máster de título «Estudio y desarrollo de modelos neuronales de complejos sistemas multivariables». En la primera parte del mismo se contextualiza el trabajo realizado, así como los objetivos y alcance del mismo.

Una vez enmarcado el proyecto de forma general, se realiza un análisis del estado del arte en el ámbito estudiado, obteniendo una visión del estado actual de este.

A continuación, se detalla el desarrollo de la solución y el trabajo realizado, y se analizan los resultados obtenidos de este, siguiendo los pasos realizados para la realización del proyecto.

Una vez presentado el trabajo desarrollado, esta memoria cuenta con una planificación de los trabajos a realizar y las relaciones entre dichas tareas. En esta sección se incluye además el presupuesto del proyecto, y se realiza el seguimiento, tanto de este, como del trabajo planificado.

Para cerrar el documento, se realizan unas conclusiones finales, donde se estudia el trabajo realizado y sus resultados, junto al seguimiento de la planificación temporal y económica del proyecto, y se comenta sobre el posible trabajo futuro a realizar.

2. Contexto

El mundo que nos rodea está repleto de sistemas, en muchos de los cuales se hace necesario ejercer una regulación sobre ellos para resultar beneficiosos, o para extraer su máximo potencial. Por esta razón, el mundo actual no puede entenderse sin la ingeniería de control, encargada de analizar y diseñar sistemas de control para realizar esta tarea de regulación, entre otras muchas funciones que tienen.

Esta disciplina, al igual que la necesidad del control, no es algo nuevo, conociéndose que existen controles simples desde la Antigua Grecia. Sin embargo, es con la máquina de vapor y la Revolución Industrial que el control empieza a ganar importancia, y desde finales del siglo XIX que empieza a desarrollarse la teoría de control.

Respecto a los sistemas de control, los más simples son los llamados controles todo/nada. Un ejemplo sencillo es un termostato, que enciende (todo), por ejemplo, una calefacción, cuando la temperatura es menor a un cierto umbral, y la apaga (nada) al alcanzarlo.

Sin embargo, estos sistemas de control suelen ofrecer baja precisión y pueden causar comportamientos oscilatorios, o incluso inestables. Es por ello, que se han desarrollado sistemas de control más complejos, en los que surge la necesidad de conocer no sólo el resultado deseado (variable controlada) sino qué variable modificar para obtenerlo (variable manipulada), además de la relación entre ambas (planta). Un control simple que sigue estos conceptos es el control de realimentación (Figura 1).

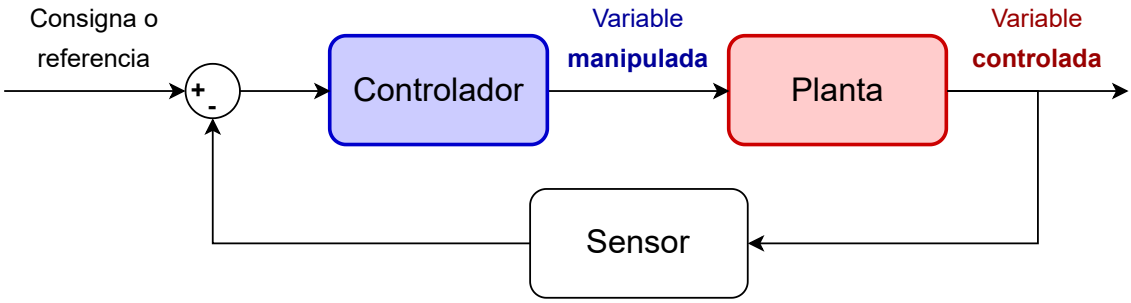

Figura 1: Esquema de un control mediante realimentación negativa

Para el diseño del controlador, es importante desarrollar un modelo de la planta. Dependiendo del sistema y el ámbito de aplicación del sistema de control, se pueden emplear modelos muy precisos, basados en el conocimiento de multitud de datos (normalmente físicos) del sistema a controlar, ya sea mediante conocimiento de las características del mismo, o mediante medidas y ensayos.

Esta alta precisión de los modelos es muchas veces necesaria por requerimientos del sistema, tales como dinámicas muy rápidas, ruido en señales, consignas rápidamente cambiantes...

También existen diversas estrategias de control basadas completamente o en mayor parte en el modelo, tales como el control predictivo (MPC). En estos casos, los errores en el modelado de la planta se propagan en gran medida, provocando que un pequeño error en el modelo pueda degradar notablemente la respuesta del sistema, por lo que requiere de modelos muy precisos.

Sin embargo, no siempre es fácil derivar este modelo ideal, debido a incertidumbres en las medidas, o simplemente, que el sistema sea complejo. Dentro de esta complejidad se incluyen, entre otros, dificultades para estimar la relación exacta entre las variables del sistema, la existencia en el sistema de múltiples entradas y salidas (sistemas MIMO), con acoplamientos entre ellas, o comportamientos no-lineales, para los cuales es más difícil desarrollar sistemas de control.

Para esta problemática, se han desarrollado diversas líneas dentro de la ingeniería de control, siendo una de ellas el empleo de modelos de caja negra. En estos modelos, a diferencia de los modelos mencionados anteriormente donde se conocen las ecuaciones del sistema (caja blanca), se desconoce el comportamiento interno del modelo.

En otras palabras, se trata de desarrollar un modelo que, a igualdad de entradas, produzca las mismas salidas que el sistema real, pero mediante operaciones desconocidas. Esto evidentemente omite la necesidad de conocer el funcionamiento interno exacto del sistema, pero crea otro problema, ya que este debe seguir reflejando el comportamiento del sistema, pero esta vez sin recurrir al conocimiento del mismo.

En este aspecto, las redes neuronales artificiales (RNA) se han alzado como excelentes modelos de caja negra [1][2]. Adicionalmente, las RNA son sistemas excepcionalmente flexibles, gracias a su capacidad de aproximación y generalización, y su multitud de parámetros adaptables para cada situación, entre ellos, su multitud de estructuras, entradas, salidas, y conexiones entre las neuronas.

De esta forma, no es de extrañar el estudio y desarrollo del uso de redes neuronales en diversas estrategias de control esté a la orden del día, especialmente para el modelado e identificación de sistemas no-lineales.

3. Objetivos y alcance

El Trabajo de Fin de Máster se ha realizado dentro del Grupo de Investigación de Control Inteligente (GICI) y del Departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingeniería de Bilbao.

El objetivo del presente Trabajo de Fin de Máster es el estudio y desarrollo de modelos neuronales de complejos sistemas multivariables, y el desarrollo de programas que faciliten el desarrollo y entrenamiento de dichos modelos.

Para este fin, se han dispuesto una serie de objetivos parciales, que permiten alcanzar el objetivo final de forma incremental.

En primer lugar, se han buscado estudiar las tipologías de RNA y las estrategias de entrenamiento utilizadas. Dicho en otras palabras, se ha realizado un análisis del estado del arte del ámbito previo al comienzo del desarrollo.

A continuación, una vez realizado el acercamiento teórico al problema, se ha realizado un acercamiento práctico, realizando modelos de sistemas simples con las herramientas a utilizar (MATLAB), para aprender a emplear las funcionalidades ofrecidas.

Tras asentar las bases teóricas y prácticas, se ha comenzado el desarrollo como tal, buscando realizar los modelos de los sistemas objetivo, comenzando por aquellos con menor grado de complejidad.

Una vez desarrollado el modelo neuronal de un sistema, se busca validarlo de forma doble. De esta forma, no sólo se busca que la red neuronal sea capaz de generalizar y evitar sobredimensionamientos y sobreentrenamientos, sino que además se ha buscado que sea capaz de predecir resultados futuros, para su empleo en estrategias de control predictivo, como iMO-NMPC [3].

Finalmente, con vistas al trabajo futuro, se ha simplificado y generalizado el código, de forma que se pueda optimizar el desarrollo de futuros modelos neuronales en la línea de investigación del grupo.

De esta forma, se obtienen dos resultados finales, ya que, por un lado, se obtienen modelos neuronales de sistemas complejos, empleables en estrategias de control que los utilicen, como el ya mencionado iMO-NMPC.

Por otro lado, se ha desarrollado un programa capaz de optimizar el desarrollo y entrenamiento de las redes, además de ayudar a establecer los paramétros de las redes, tales como número de neuronas, estructuras de entradas y salidas, realimentaciones... de forma que el modelo neuronal aproxime el sistema de forma precisa, pero minimizando el coste computacional de la misma, con vistas a su implementación en sistemas reales.

Con el fin de realizar una primera estimación del plazo y presupuesto del proyecto, se ha realizado una planificación inicial. Esta planificación incluye las fases a seguir para el desarrollo del proyecto, y su duración estimada.

Se estima que el proyecto comience el lunes, 23 de mayo de 2022. El horario de trabajo es de lunes a viernes, 8 horas al día, a excepción de las dos primeras semanas de agosto, que se han establecido como no laborables. De esta forma, la planificación inicial del proyecto es la que se puede observar en la Tabla 1.

Fase	Nombre de fase	Duración	Comienzo	Fin
F.1	**Revisión de la propuesta del TFM**	**3 días**	**23/05/2022**	**25/05/2022**
F.2	**Estudio de estado del arte y herramientas**	**24 días**	**26/05/2022**	**28/06/2022**
F.3	**Desarrollo del trabajo**	**37 días**	**29/06/2022**	**01/09/2022**
F.3.1	**Desarrollo del código**	**15 días**	**29/06/2022**	**19/07/2022**
H1	Código desarrollado	0 días	19/07/2022	19/07/2022
F.3.2	**Desarrollo de modelos neuronales**	**15 días**	**11/07/2022**	**23/08/2022**
H2	Modelos neuronales desarrollados	0 días	23/08/2022	23/08/2022
F.3.3	**Simplificación y generalización del código**	**7 días**	**22/08/2022**	**01/09/2022**
H3	Código simplificado	0 días	01/09/2022	01/09/2022
F.4	**Documentación**	**64 días**	**23/05/2022**	**01/09/2022**

Tabla 1: Planificación inicial del proyecto

En la Tabla 1 también se puede observar que se han introducido varios hitos, que se han detallado en la Tabla 2.

Hito	Descripción	Fecha
H1	Código de generación de modelos neuronales completado	Semana 9
H2	Modelos neuronales de los sistemas objetivo generados	Semana 12
H3	Código simplificado y generalizado: proyecto finalizado	Semana 13

Tabla 2: Hitos iniciales del proyecto

Respecto al presupuesto inicial estimado, se han realizado varias consideraciones para el cálculo del mismo. Para realizar una aproximación del coste de las horas internas, se ha estimado la tasa horaria del alumno como la de un ingeniero júnior (20 €/h), y que este trabaja, como se ha comentado, durante 8 horas diarias, durante toda la duración del proyecto.

Respecto a las amortizaciones, antes de proceder al cálculo del presupuesto, se han calculado las tasas horarias de los elementos que pueden ser utilizados en otros proyectos (Tabla 3).

Para este fin, se ha estimado que la vida útil del *hardware* (PC) es de 5 años, debido al uso intensivo y continuado que se le da, mientras que el software (MATLAB y *toolboxes*) tiene un periodo de amortización de 3 años. Este menor periodo de amortización se debe a que, tras estos 3 años, se compra una licencia nueva para estar al día con las posibles actualizaciones y mejoras.

Por otro lado, se han estimado los costes indirectos en 4 % adicional sobre los costes directos, se ha añadido una partida del 10 % de este subtotal para posibles imprevistos, y se ha estimado el coste de capital en el 2 %.

De esta forma, el presupuesto estimado es de 12.695,65 € (Tabla 3).

Cálculo de tasas horarias			
	Coste (€)	Vida útil (h)	Tasa horaria (€/h)
PC	2.000	9.000	0,222
MATLAB R2022a	2.100	5.400	0,389
Deep Learning Toolbox	1.200	5.400	0,222
Parallel Computing Toolbox	1.050	5.400	0,194
Control System Toolbox	1.200	5.400	0,222

HORAS INTERNAS	Cantidad (h)	Tasa horaria (€/h)	Coste (€)	**10.240,00 €**
Alumno	512	20	10.240,00	
AMORTIZACIONES	Cantidad (h)	Tasa horaria (€/h)	Coste (€)	**640,00 €**
PC	512	0,222	113,78	
MATLAB R2022a	512	0,389	199,11	
Deep Learning Toolbox	512	0,222	113,78	
Parallel Computing Toolbox	512	0,194	99,56	
Control System Toolbox	512	0,222	113,78	
COSTES DIRECTOS				**10.880,00 €**
Costes indirectos	4 %		435,20 €	
SUBTOTAL				**11.315,20 €**
Imprevistos	10 %		1.131,52 €	
SUBTOTAL 2				**12.446,72 €**
Coste de capital	2 %		248,93 €	
TOTAL				**12.695,65 €**

Tabla 3: Tasas horarias y presupuesto estimado del proyecto

4. Estado del arte

Una vez enmarcado el contexto, objetivos, y alcance del trabajo, en este apartado se incluyen los detalles de más bajo nivel necesarios para comprender el problema atajado. Además, este apartado incluye los antecedentes bibliográficos dentro del ámbito trabajado.

De esta forma, el apartado se divide en tres secciones, cada una bajando a un nivel progresivamente más bajo del problema, especificando el punto de desarrollo concreto dentro del ámbito que se ha atajado en este trabajo, así enmarcando completamente el desarrollo de la solución realizado.

4.1. Redes neuronales (RNA)

Tal y como se ha comentado anteriormente, el objetivo del trabajo realizado es el desarrollo de modelos neuronales para identificación y modelado de sistemas. Estos son modelos de caja negra, donde se busca modelar el sistema desconociendo el funcionamiento interno del mismo, en este caso, mediante el empleo de redes neuronales artificiales (RNA).

Las redes neuronales artificiales (RNA) son sistemas computaciones compuestos por diversos nodos llamados *neuronas artificiales*, o simplemente *neuronas*, conectadas direccionalmente entre sí; de esta forma, las redes neuronales reciben unos valores de entrada, y mediante diversas operaciones realizadas en cada neurona, y las conexiones entre estas, produce unos valores de salida.

Para esto, cada neurona recibe las señales de todas las neuronas conectadas a ella, realizando la suma ponderada de todas ellas, denominándose la ponderación de cada una de las señales de entrada *peso* (w). A esta suma se le añade el denominado *bias* de la neurona (b), que en otras palabras es el valor de la suma en el caso de que todas las señales de entrada sean nulas.

El valor de esta suma no sale directamente de la neurona, sino que se le aplica una operación denominada *función de activación* $s(x)$. Al comienzo del desarrollo de neuronas artificiales y RNA, esta función era comúnmente la función umbral, una función binaria por la que, si la suma ponderada superaba un cierto valor umbral, la neurona produce un valor alto (normalmente 1), y en caso de no superar este valor, produce un valor bajo (normalmente 0 o -1).

De esta forma, la salida de una neurona (y) con m entradas (x) se calcula de la siguiente forma:

$$y = s(b + \sum_{k=1}^{m} w_k x_k)$$

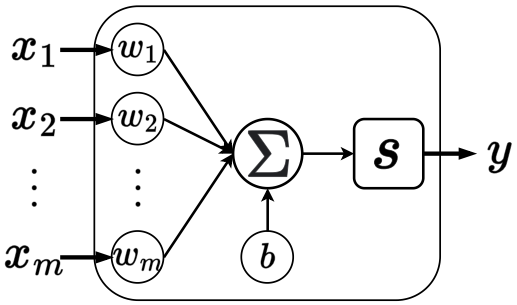

Figura 2: Esquema de una neurona artificial

Las redes neuronales se pueden clasificar según distintos criterios, siendo uno de los principales la clasificación de las redes según sus conexiones, donde, aunque hay más tipos, se dividen en dos grandes bloques: las redes *prealimentadas* o *feed-forward* y las redes *recurrentes* (Figura 3), siendo un desarrollo posterior.

La diferencia entre ambas radica en que en las redes prealimentadas, las conexiones entre neuronas no forman un ciclo, es decir, la transferencia de señales o valores es unidireccional; mientras que en las redes recurrentes, las conexiones sí forman un ciclo.

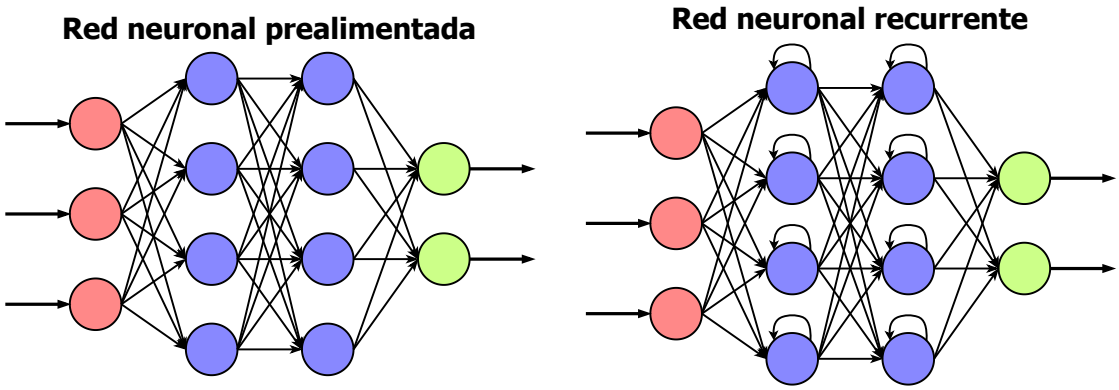

Figura 3: Redes neuronales según sus conexiones

Aunque las estructuras de las redes han evolucionado con el tiempo para afrontar diversos problemas, uno de los mayores avances en el desarrollo de redes neuronales ha sido el empleo de funciones de activación derivables y no-lineales, ya que incrementó su funcionalidad en enorme medida, especialmente el de las redes prealimentadas.

La primera de ellas, porque permitía aplicar en redes prealimentadas el algoritmo de entrenamiento de *propagación hacia atrás* (*backpropagation*), permitiendo así el entrenamiento de redes neuronales de múltiples capas.

Por otro lado, fue demostrado [4] que, cuando la función de activación de las neuronas es no-lineal, las redes neuronales prealimentadas, incluso de únicamente dos capas (más la capa de entrada), son capaces de ejercer como aproximadores universales de funciones [5]; en lo que ha pasado a ser conocido como «Teorema de Aproximación Universal».

Son estas características las que poseen la mayoría de funciones de activación empleadas en la actualidad, especialmente las *funciones de activación sigmoidales*, como la función logística $\sigma(x) = \dfrac{1}{1+e^{-x}}$, o la función tangente hiperbólica $\tanh(x) = \dfrac{e^{x} - e^{-x}}{e^{-x} + e^{-x}}$.

Estas funciones se denominan *sigmoidales* porque la relación entre entradas y salidas de estas funciones tienen forma de S (Figura 4), es decir, son monótonamente crecientes y poseen dos asíntotas horizontales.

La diferencia más notable entre ambas es el rango de salidas de la función: la función logística produce valores entre 0 y 1, y la función tangente hiperbólica produce valores entre -1 y 1. También puede comprobarse que la función tangente hiperbólica realiza un cambio más brusco de la salida (Figura 4), alcanzando sus valores límite de y para valores de x más cercanos a 0 que la función logística.

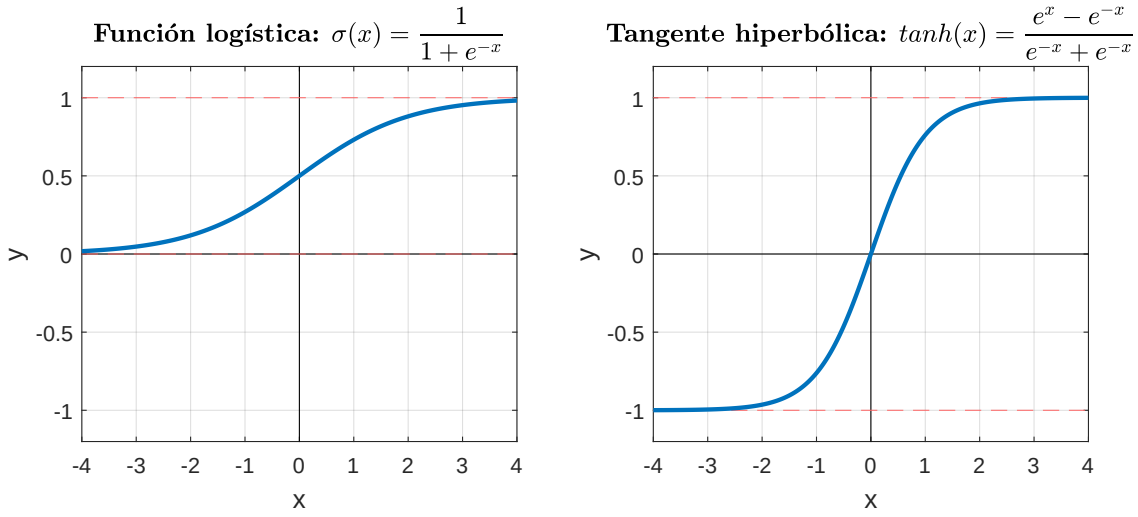

Figura 4: Funciones de activación sigmoidales

Para ser capaz de producir los resultados deseados, todos los pesos y *bias* de la red neuronal, denominados *parámetros de la red*, deben ajustarse de forma precisa. A este proceso de ajuste de los parámetros de la red se le denomina *entrenamiento* o *aprendizaje*, ya que es el proceso encargado de que la red *aprenda* la relación entre las entradas y salidas de la misma.

Existen tres paradigmas distintos para el entrenamiento de redes neuronales, generando redes neuronales cuyo objetivo es diferente:

- Entrenamiento no supervisado: En este caso, para el entrenamiento, la red recibe únicamente las entradas de la red, siendo el objetivo de la red la de inferir patrones entre las entradas y salidas predichas, por lo que suelen ser empleadas para usos de agrupación de datos (*data clustering*).
- Entrenamiento por refuerzo: Para el entrenamiento, la red recibe las entradas de la red, y una función de coste a minimizar (o función de refuerzo o recompensa a maximizar). De esta forma, aunque no se dan las salidas, la red es capaz de conocer si las salidas predichas son correctas o no, y adaptarse a ello. Actualmente este es un campo de interés para diversas aplicaciones, entre ellas, en sistemas de control, para el desarrollo de controladores [6].
- Entrenamiento supervisado: A diferencia de los paradigmas anteriores, en este caso, para el entrenamiento se emplean tanto las entradas, como las salidas exactas deseadas de la red. De esta forma, el objetivo del entrenamiento es que la red produzca unas salidas lo más cercanas a las deseadas. Las tareas entrenadas de esta forma son, entre otras, la clasificación de datos, y la aproximación de funciones.

Es dentro de esta última categoría donde se sitúa el trabajo realizado, ya que el objetivo del mismo es el desarrollo de modelos de sistemas, que en última instancia, se pueden interpretar como funciones que, con una serie de entradas, producen una serie de salidas.

El algoritmo más ampliamente empleado para realizar el entrenamiento supervisado de redes neuronales prealimentadas es la *propagación hacia atrás* (*backpropagation*), mencionado anteriormente.

Este algoritmo computa el gradiente de la función de coste, como por ejemplo, el error entre la salida predicha por la red ($\hat{y}$) y la salida real deseada (y); respecto a los parámetros de la red.

Con estos gradientes, el algoritmo busca encontrar el mínimo de la función de coste, y por tanto, conseguir que la salida predicha de la red se asemeje más a la deseada. Es de este proceso de donde coge el nombre el algoritmo, ya que propaga los errores (de salida) hacia atrás (hacia los parámetros de red).

Sin embargo, una de las limitaciones de este algoritmo de entrenamiento es que no asegura que al final del mismo, los valores converjan al mínimo global de la función, sino únicamente a un mínimo local. Aunque esto puede no ser un impedimento en ocasiones, es común realizar varios entrenamientos, inicializando los parámetros de la red a distintos valores, explorando así la hipersuperficie del error, con el objetivo de encontrar el mínimo global, o un mínimo local lo más cercano posible.

Dentro de este algoritmo de entrenamiento, existen distintas variaciones, siendo muy utilizado el *algoritmo de Levenberg-Marquardt* para el entrenamiento [7]. Este algoritmo modifica el algoritmo original, empleando derivadas de segundo orden, consiguiendo que el error converja más rápidamente, reduciendo el número de iteraciones necesarias para realizar el entrenamiento.

Como último punto del desarrollo de una red neuronal, se encuentra la validación. El objetivo de la validación es comprobar que la red neuronal haga, en efecto, la tarea para la que ha sido diseñada. Esto es debido a que, aunque durante el entrenamiento se minimice el error cometido, cabe recordar que este error es el cometido respecto a los datos de entrenamiento empleados.

Para ser funcional, la red neuronal debe poseer la capacidad de *generalización*, es decir, debe ser capaz de realizar la tarea diseñada para cualquier entrada, por supuesto, para el rango para el que ha sido entrenada; pero en ocasiones para valores fuera de este rango, lo que se denomina que la red posea capacidad de *extrapolación*.

Un problema que puede ocurrir durante el entrenamiento es el *sobreajuste* (*overfitting*) de la red. En esta situación, la red neuronal aprende *demasiado* bien los datos de entrenamiento, y pierde la capacidad de generalización.

Es para comprobar esto, uno de los métodos más empleados es emplear la red para un conjunto distinto de datos, llamado conjunto de datos de validación, y calcular alguna medida de error entre los valores producidos por la red, y la salida real de esta.

Muchas veces este proceso se realiza a la par del entrenamiento, donde tras cada iteración del entrenamiento, se realiza la validación de la red. Durante un entrenamiento correcto, el error cometido en la validación se debe reducir en cada iteración.

Si tras alguna iteración este error aumenta, puede ser un signo de sobreajuste (Figura 5), siendo posible ejecutar distintas acciones. Una de las acciones más comunes es dejar un número de iteraciones de margen, esperar si el error vuelve a disminuir, y si no lo hace, tomar los parámetros de red de la iteración con menor error de validación.

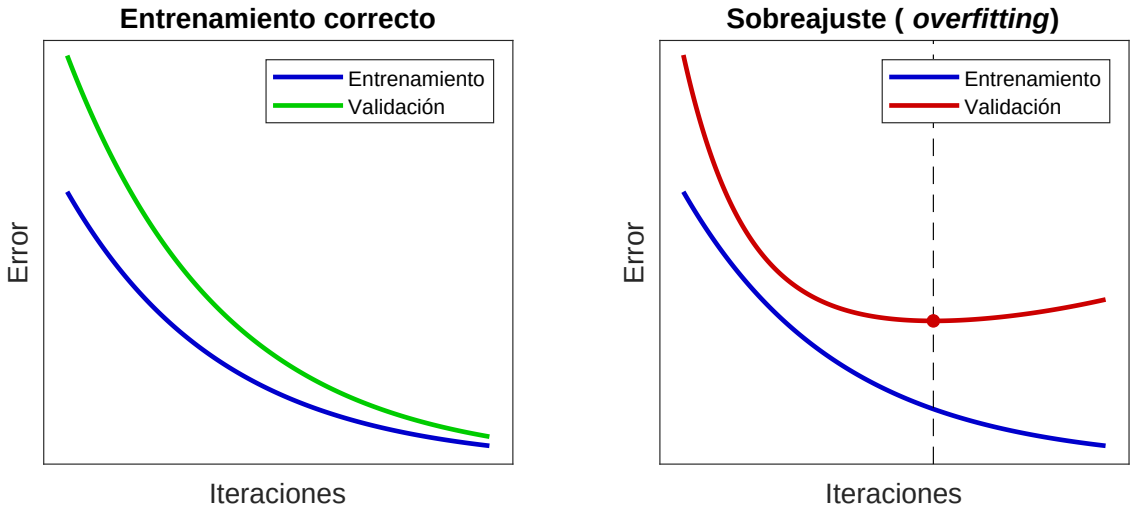

Figura 5: Comparación entre un entrenamiento correcto y una situación de sobreajuste

4.2. Identificación y modelización de sistemas mediante RNA

Tal y como se ha explicado en la sección anterior, las redes neuronales artificiales se pueden usar en multitud de aplicaciones, como tareas de clasificación, en casos como procesamiento de imágenes o reconocimiento de voz; procesamiento del lenguaje natural (*natural language processing*), controladores en sistemas de control [6], inferencia estadística, búsqueda de patrones y, el uso realizado en este trabajo, como modelos de sistemas.

Así, la tarea desarrollada entra dentro del ámbito de la aproximación de funciones, uno de los usos más extendidos de las RNA; más concretamente dentro del modelado de series temporales *time-series modelling*.

Como se ha mencionado anteriormente, las redes neuronales prealimentadas (*feed-forward*) son capaces de ejercer como aproximadores universales de funciones, si cumplen tres condiciones:

- La red neuronal posee al menos dos capas, más la capa de entrada, que no se suele contar para el cómputo del número de capas de la red.
- Las neuronas de las capas intermedias emplean una función de activación no lineal, como las funciones sigmoidales.
- La capa de salida es lineal, es decir, las neuronas de la capa de salida emplean funciones de activación lineales, como $f(x) = x$.

Por esta razón, para la realización de este trabajo se han estudiado modelos neuronales de sistemas mediante el empleo de redes neuronales prealimentadas que cumplen estas condiciones.

Dentro de las redes neuronales prealimentadas, la más común es el *perceptrón multicapa* (*multi-layer perceptron*, MLP), compuestas por múltiples capas de neuronas, estando todas las capas *totalmente conectadas*, es decir, cada neurona de una capa conecta con *todas* las neuronas de la capa siguiente.

Las capas de los perceptrones multicapa se dividen en tres secciones (Figura 6): la capa de entrada, encargada de recibir y procesar las señales de entrada; las capas ocultas, encargadas de procesar las datos mediante una o más capas de neuronas; y la capa de salida, que produce el resultado de salida de la red.

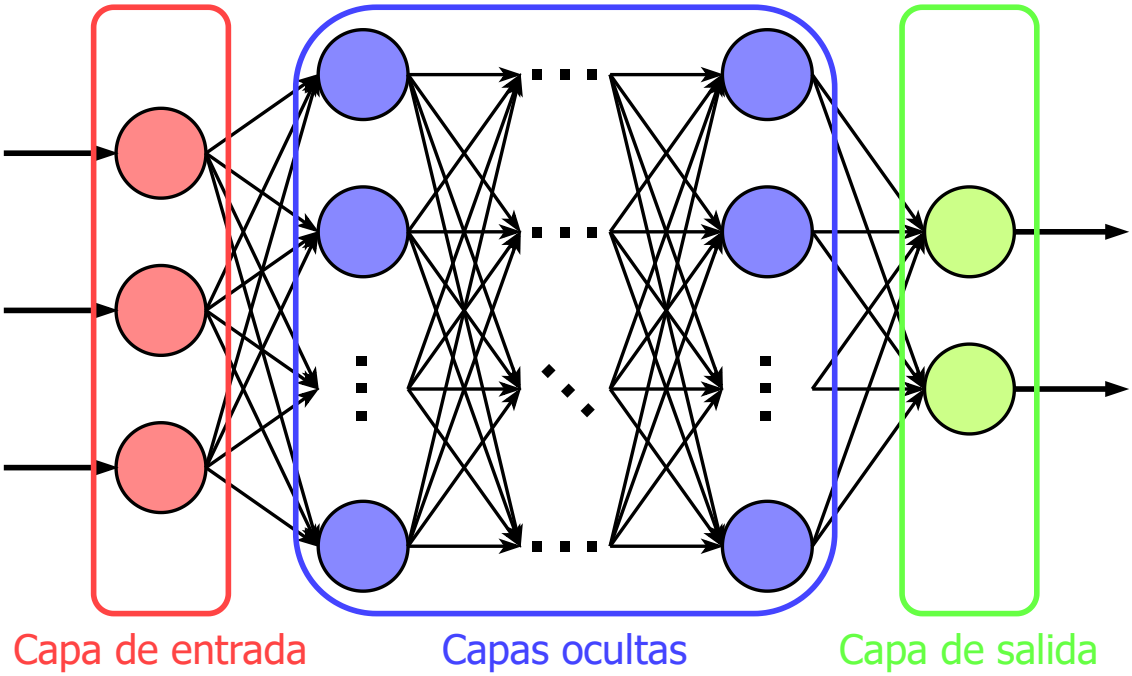

Figura 6: Esquema de un perceptrón multicapa (MLP)

Los perceptrones multicapa pueden clasificarse según su número de capas ocultas. Los MLP con más de una capa oculta se denominan *redes neuronales profundas* (*deep neural network*), mientras que las redes con una única capa oculta, en contraste, se las denomina *redes neuronales no-profundas* (*shallow neural network*).

Debido a que el Teorema de Aproximación Universal asegura que los MLP con una única capa oculta (ya que requiere dos capas, una capa oculta y la de salida), es decir, MLP no-profundos, son capaces de realizar el trabajo deseado, son estas redes las que se han empleado.

La mayor restricción de los MLP son su poca capacidad de extrapolación, por lo que los resultados obtenidos de datos fuera del rango de datos empleado en el entrenamiento pueden tener bastante error. Es por este motivo que es importante estudiar bien el rango de valores en el que va a trabajar el sistema a modelar, y entrenar la red con entradas que representen bien este rango de valores, especialmente hacia los límites.

Esto, unido a la limitación en la convergencia del algoritmo de entrenamiento de propagación hacia atrás, que puede hacer que los parámetros de la red converjan a un mínimo local, tal y como se ha explicado anteriormente; hace que se deba prestar atención al diseño del entrenamiento y de los datos datos empleados en él, ya que ambas limitaciones son eludibles hasta cierto punto, con un diseño adecuado.

Una vez establecida la elección de la familia de redes neuronales a emplear, en este caso, perceptrones multicapa no-profundos, se debe elegir cual será el conjunto de entradas de la red empleado, al que se denomina *vector de regresión*, ya que las variables empleadas como entradas se denominan *regresores*. Existen distintas estructuras de redes neuronales según el vector de regresión empleado. Las variables que se pueden emplear como regresores son:

- Entradas externas o exógenas actuales y pasadas a la red (u)
- Salidas pasadas reales (medidas) del sistema (y)
- Salidas pasadas calculadas por la red (salidas predichas) ($\hat{y}$)
- Error de salida pasado calculado ($e_y = y - \hat{y}$)

Cabe destacar que las dos últimas dependen de la salida de la propia red, por lo que si se emplean como regresores, la red efectivamente funcionaría como una red recurrente, ya que existiría un ciclo en las conexiones; aunque estrictamente hablando, la red seguiría siendo prealimentada, debido a que entre la capa de entrada y salida no hay ciclos, este existe fuera de la red.

La primera y más simple de las estructuras MLP es la estructura NNFIR (*Neural Network Finite Impulse Response*, red neuronal de respuesta finita al impulso), empleada como predictor FIR no-lineal, por lo que posee el mismo vector de regresión que estos modelos. Los modelos FIR se representan mediante la siguiente ecuación, donde q^{-1} representa un retardo unitario:

$$y(k) = B(q)u(k) + e(k) = b_0u(k) + b_1u(k-1) + \cdots + b_nu(k-n) + e(k)$$

De esta forma, el predictor, en este caso la red neuronal (Figura 7), sigue la ecuación $\hat{y}(k|\theta) = B(q)u(k)$, donde θ se refiere a los parámetros de la red neuronal, ya que la salida depende de ella; y por tanto, según la ecuación el vector de regresión $\varphi_{FIR}(k)$ es:

$$\varphi_{FIR}(k) = [u(k), u(k-1), u(k-2), \ldots, u(k-n)]^T$$

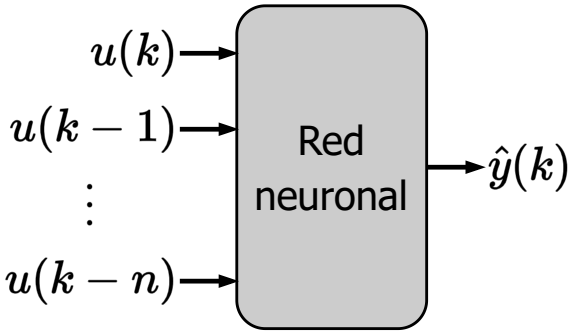

Figura 7: Red neuronal NNFIR

De esta forma, las entradas de la red son únicamente las entradas exógenas del sistema u. Este predictor, y por tanto, esta estructura de red neuronal, aunque es siempre estable, presenta un problema, ya que muchos sistemas dinámicos, incluidos los trabajados en este proyecto, necesitan información de los estados pasados de la salida, y por tanto, no es capaz de modelar con precisión el sistema deseado, incluso introduciendo un número elevado de muestras de la entrada u.

Esto puede parecer contradecir al Teorema de Aproximación Universal, ya que se cumplen todas las condiciones y, sin embargo, este modelo no es viable para el trabajo a realizar. En este caso, el detalle radica en que este teorema asegura que se puede aproximar cualquier función, pero no especifica el número de neuronas necesario.

Para este caso, debido a la problemática con los sistemas dinámicos, este número de neuronas tendría que ser muy elevado, y por tanto, la red sería computacionalmente inviable para el trabajo realizado, ya que se desea emplear estos modelos en sistemas de control en tiempo real.

Una solución a esta problemática pasa por introducir las predicciones pasadas de la red como entradas, creando así la estructura NNOE (*Neural Network Output Error*, red neuronal de error de salida). Los modelos OE se representan mediante una ecuación similar a la de los modelos FIR:

$$y(k) = \frac{B(q)}{F(q)}u(k) + e(k)$$

$$F(q) = 1 + f_1 q^{-1} + f_1 q^{-1} + f_2 q^{-2} + \cdots + f_{n_y} q^{-n_y}$$

Así, en estos modelos, el predictor sigue la ecuación $\hat{y}(k|\theta) = \frac{B(q)}{F(q)}u(k)$. Operando, puede comprobarse fácilmente que, efectivamente, la salida predicha depende de sus predicciones anteriores $\hat{y}(k)$, siendo así su vector de regresión $\varphi_{OE}(k)$:

$$\varphi_{OE}(k) = [u(k), u(k-1), u(k-2), \ldots, u(k-n), \hat{y}(k-1), \ldots, \hat{y}(k-n_y)]^T$$

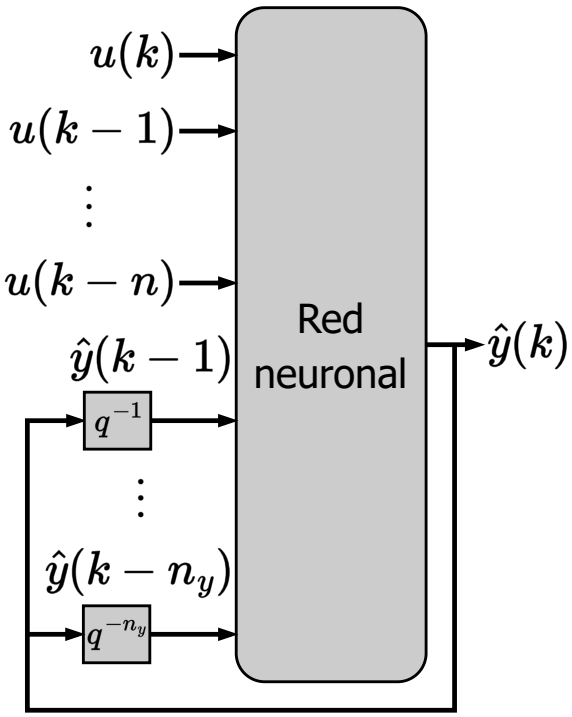

Figura 8: Red neuronal NNOE

Sin embargo, tal y como se ha comentado anteriormente, y se observa en la Figura 8, estas redes neuronales introducen una realimentación de la salida, convirtiéndolas en redes recurrentes.

La gran desventaja de estos modelos es que, debido a la realimentación, es más difícil reducir el error durante el entrenamiento, debido a un cálculo más complejo de los gradientes de errores.

Adicionalmente, la estabilidad de este predictor depende de $F(q)$, que depende de los parámetros de la red, por lo que debe ser monitorizada durante el entrenamiento para evitar que el predictor se inestabilice [8].

Una estructura que evita los problemas de los dos anteriores es la denominada estructura NNARX (*Neural Network AutoRegressive eXogenous*, red neuronal autorregresiva exógena). Esta es otra modificación sobre los modelos FIR, pero a diferencia de los modelos OE, donde se emplean las salidas predichas $\hat{y}$, se emplean las salidas reales medidas del sistema y. De esta forma, los modelos ARX se representan mediante la siguiente ecuación:

$$A(q)y(k) = B(q)u(k) + e(k)$$

Mediante esta modificación, el predictor evita emplear sus propias salidas, evitando crear ningún tipo de realimentación. De esta forma, la ecuación del predictor es la siguiente:

$$\hat{y}(k) = b_0u(k) + b_1u(k-1) + \cdots + b_nu(k-n) - a_1y(k-1) - \cdots - a_{n_y}y(k-n_y)$$

Tal y como se ve en la ecuación, se puede también decir que el vector de regresión $\varphi_{ARX}(k)$ en este caso es:

$$\varphi_{ARX}(k) = [u(k), u(k-1), u(k-2), \ldots, u(k-n), y(k-1), \ldots, y(k-n_y)]^T$$

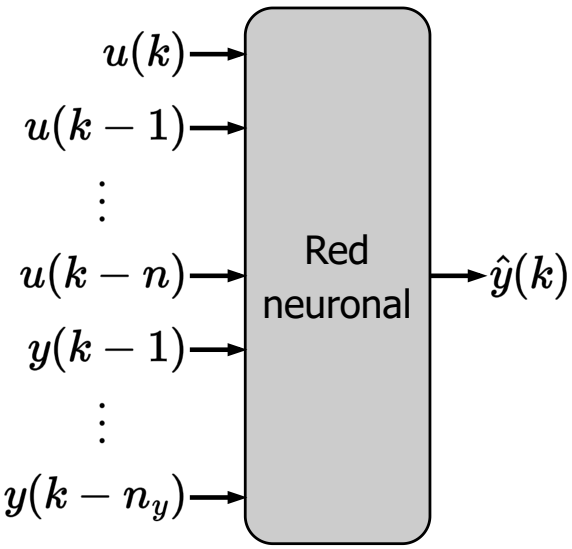

Figura 9: Red neuronal NNARX

Mediante este modelo, es posible realizar la identificación de sistemas dinámicos como los modelados en este trabajo; a la vez que se evitan los problemas derivados de la realimentación de las predicciones. Adicionalmente, estos predictores son siempre estables.

Por otro lado, una de las mayores limitaciones que presentan los predictores NNARX es derivada de la necesidad de utilizar las salidas reales, es decir, se presenta en la obtención y medición de estos datos.

Gracias a las ventajas ofrecidas por este tipo de modelos, es posible afrontar la identificación de los sistemas deseados. Además, en este trabajo se emplean, como se detalla más adelante, sistemas sintéticos en simulación, por lo que en estos casos, esta problemática puede obviarse.

De esta forma, las redes neuronales con estructura NNARX poseen múltiples cualidades que hace que se ajusten a las necesidades del proyecto:

- Capaces de modelar las dinámicas de los sistemas empleados con un número de neuronas reducido, a diferencia de los predictores FIR.
- Los predictores son estables.
- El entrenamiento es más simple que el de los predictores NNOE, debido a que no necesitan de realimentación de las salidas predichas $\hat{y}$.
- Eluden la mayor limitación debido a su uso en simulación, donde la obtención de las salidas reales no presenta ningún problema.

Son todas estas las razones, de forma combinada, por las que se ha empleado esta estructura para el desarrollo del trabajo.

4.3. NNARX en identificación de sistemas

Una vez elegida la estructura de la red, se ha realizado un análisis del empleo de estas redes para la identificación de diversos sistemas, mediante una búsqueda de antecedentes bibliográficos.

Durante esta búsqueda, se han encontrado empleos de redes neuronales NNARX, o simplemente NARX (*Nonlinear AutoRegressive eXogenous*), ya que son el modelo que implementan; para la identificación de modelos de diversas índoles.

De esta forma, son muchos ámbitos en los que se ha empleado esta estructura de red neuronal como modelo de sistemas, como por ejemplo, en el ámbito de la mecánica, y más concretamente, en el ámbito de las vibraciones. Ejemplos del uso de redes NARX incluyen la identificación de un modelo inverso para el control de vibración de una estructura en voladizo [9], o en el modelado de un amortiguador magneto-reológico [10], un tipo de amortiguador variable, de forma que el modelo puede ser usado para el control de la dureza del mismo.

Otro ámbito relevante, especialmente hoy en día, es el de la energía, especialmente, la energía renovable. En este ámbito, estas redes también se han empleado para la identificación de un modelo de la radiación solar directa diaria, para su predicción [11]. Este trabajo también resulta interesante porque implementa un re-entrenamiento periódico para refinar las capacidades de la red; lo que no se aplica al trabajo actual, pero se trata de una propuesta relevante para la línea de investigación.

Más cercano a la aplicación que se va a hacer, dentro del control predictivo de procesos, es el modelado de un sistema de tres tanques en cascada para control predictivo [12]. En estos dos últimos trabajos, la aplicación de los modelos obtenidos es equiparable a la del presente trabajo. En estos casos, se busca emplear los modelos para predicción.

Hablando del ámbito en que se enmarca este trabajo, como se ha comentado en apartados anteriores, el trabajo se enmarca dentro de la línea de investigación iMO-NMPC del Grupo de Investigación de Control Inteligente (GICI), adscrito a la UPV/EHU. El sistema de control propuesto [3] se trata de un control NMPC (*Nonlinear Model Predictive Control*), variante no-lineal del MPC, basado en técnicas de control inteligente.

Este sistema consta de tres partes fundamentales (Figura 10): un algoritmo genético de optimización multi-objetivo encargado de buscar las acciones de control óptimas, el *Decision Maker* basado en lógica borrosa (*fuzzy logic*) encargado de seleccionar la mejor acción de control entre las óptimas, y finalmente, en el centro del sistema, se sitúa un modelo neuronal de estructura NARX.

De esta forma, el trabajo se enmarca dentro del desarrollo de estos modelos neuronales para iMO-NMPC, dando los primeros pasos para un acercamiento sistemático al modelado de sistemas complejos para su implementación en este esquema de control.

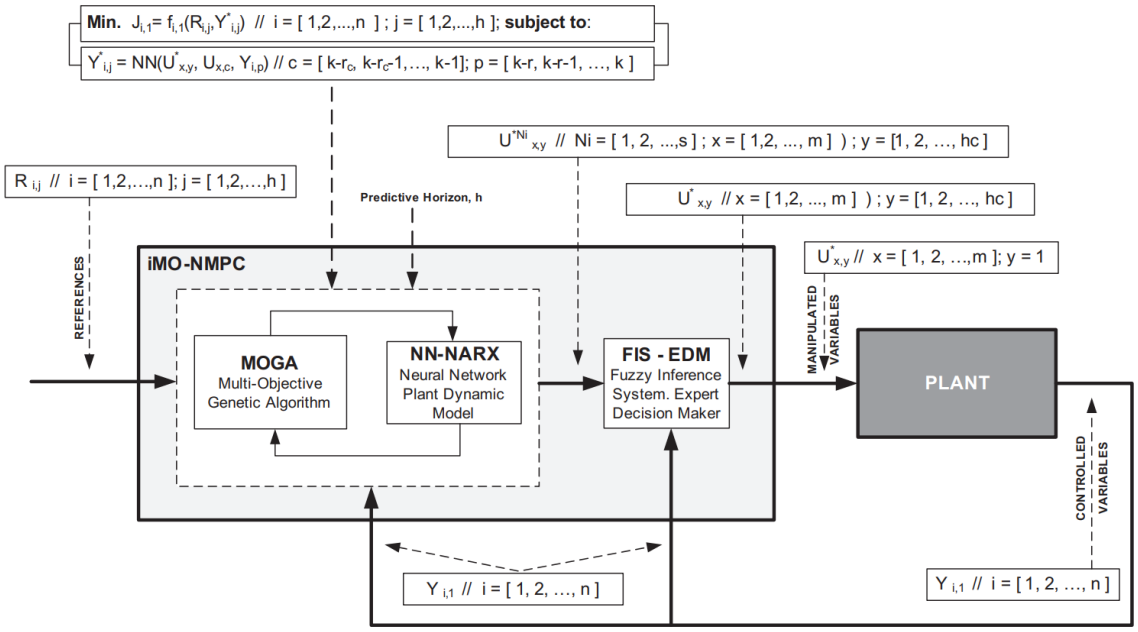

Figura 10: Estructura del iMO-NMPC [3]

Tal y como se ha especificado, para esta aplicación, los modelos van a emplearse para predicción. Por esta razón, a la hora de calcular salidas futuras, no se poseen los datos de salidas reales, ya que en realidad, todavía no han ocurrido.

La solución a esto es emplear los datos de las predicciones $\hat{y}$ en los casos donde no se poseen salidas reales y, empleándolas siempre que se tengan. A esta estructura se le llama NARX en lazo cerrado, y es la que se busca emplear en la aplicación del modelo.

Por ejemplo, en una red que emplea como regresores la muestra actual de la entrada exógena y las dos anteriores, y las tres últimas muestras de la salida real:

$$\varphi(k) = [u(k), u(k-1), u(k-2), y(k-1), y(k-2), y(k-3)]^T$$

$$\begin{aligned}
\varphi(k+1) &= [u(k+1), u(k), u(k-1), \hat{y}(k), y(k-1), y(k-2)]^T \\
\varphi(k+2) &= [u(k+2), u(k+1), u(k), \hat{y}(k+1), \hat{y}(k), y(k-1)]^T \\
\varphi(k+3) &= [u(k+3), u(k+2), u(k+1), \hat{y}(k+2), \hat{y}(k+1), \hat{y}(k)]^T
\end{aligned}$$

Esta estructura es un punto medio entre las redes NNARX y las redes NNOE, ya que no emplean únicamente salidas reales y como las primeras, pero tampoco se basan por completo en salidas predichas $\hat{y}$ como las segundas.

Es por esta razón, que en el presente trabajo se han diseñado y creado modelos neuronales NARX estándar, es decir, en lazo abierto; pero habiendo que validarlos en lazo cerrado para comprobar su efectividad en la aplicación objetivo.

Finalmente, también debido al uso que se le desea dar al modelo neuronal, se emplea un entrenamiento *off-line*, es decir, el modelo se entrena y valida, y durante la aplicación, los parámetros de la red (pesos y *bias*) de la misma permanecen constantes.

5. Análisis de riesgos

Antes de comenzar el desarrollo del proyecto, se ha realizado un análisis de los posibles riesgos que pueden darse a lo largo de este. De esta forma, tras identificar los riesgos que presentan una mayor amenaza potencial para el desarrollo del trabajo, se ha planteado una solución a estas situaciones con el objetivo de minimizar los posibles riesgos.

Para realizar este análisis, el primer paso es, como se ha dicho, identificar los riesgos existentes en el proyecto, tras lo que se ha realizado un estudio del impacto que tendrían en el caso de que se dieran durante el proyecto, y la probabilidad de que ocurran.

Con estas estimaciones, se ha realizado una matriz de probabilidad-impacto (Figura 11), mediante la que se han determinado cuáles son las situaciones que entrañan un mayor riesgo al proyecto. Es sobre estas acciones sobre las que se han planteado acciones a realizar para minimizar su impacto en el proyecto, o al menos, estar preparados con antelación.

De esta forma, los riesgos identificados son los siguientes:

A. Error de conexión con el servidor: Para el desarrollo del proyecto, se han empleado las herramientas del entorno de MATLAB donde, para verificar la licencia de uso, se requiere de una conexión a internet. En caso de pérdida de conexión, o de error en la misma, no sería posible utilizar las herramientas, por lo que no sería posible avanzar en la mayoría de tareas del proyecto.

   La probabilidad de que esto ocurra es moderada, pero su impacto es reducido, ya que es un problema sencillo de minimizar, revisando la conexión del ordenador o, en última instancia, contactando con el servicio técnico; por lo que únicamente causaría pequeños retrasos.

B. Pérdida de la memoria de almacenamiento: El trabajo realizado se guarda en el almacenamiento de un ordenador (disco duro). En caso de que ocurra algo que haga perder estos datos, se perdería todo el progreso realizado hasta el momento, lo que resulta en un gran retraso.

   La probabilidad de que esto ocurra es baja, pero su impacto sería elevado.

C. Inviabilidad computacional del resultado: Como se ha comentado, el objetivo del trabajo es el desarrollo de modelos neuronales para el empleo en esquemas de control. Para este fin, los modelos deben tener un coste computacional lo suficientemente reducido para su implementación en sistemas de control en tiempo real. En el caso de generar modelos demasiado computacionalmente intensivos, puede que sea imposible realizar esta implementación.

   La probabilidad de que esto ocurra es moderada, sin embargo, el impacto sería muy elevado, ya que, aunque el trabajo no perdería su valor teórico, no realizaría muchos avances hacia esta deseada implementación práctica.

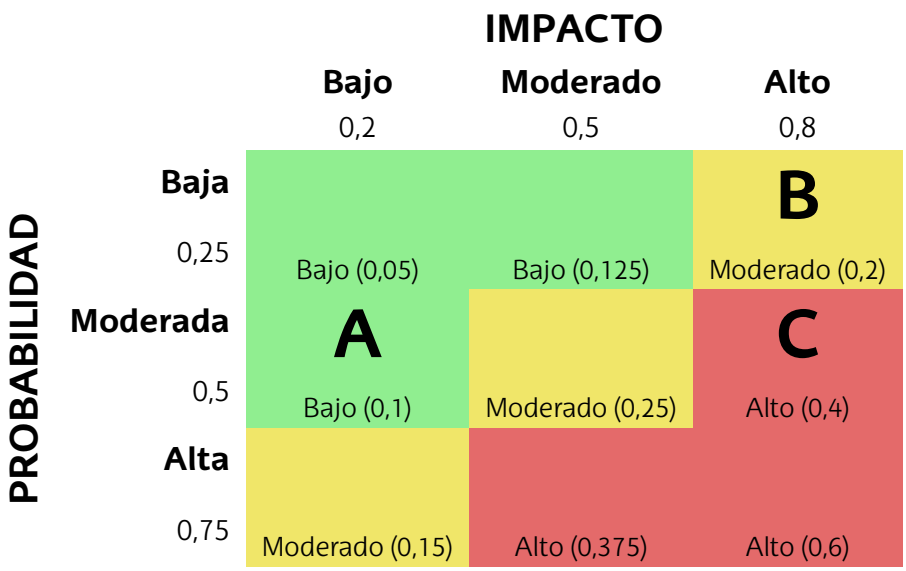

Figura 11: Matriz probabilidad-impacto de los riesgos del proyecto

De esta forma, como se puede observar en la matriz de probabilidad-impacto (Figura 11), las situaciones sobre las que se deben tomar acciones son los riesgos B y C, teniendo la situación B riesgo moderado, y la situación C, riesgo elevado. Por ello, las acciones tomadas para minimizar estos riesgos son:

B. Para no perder progeso del proyecto, se realizarán copias de seguridad en la nube cada 3 días. Así, en caso de perder los datos del disco duro, el máximo retraso sería de 3 días, lo que es un retraso aceptable.

C. Para reducir el coste computacional de los modelos, se dará un mayor peso a este aspecto a la hora de seleccionar el modelo neuronal. Adicionalmente, se buscarán maneras de reducir el coste computacional de otras maneras, como la implementación de algoritmos más eficientes para el cálculo de salidas de la red neuronal.

6. Desarrollo de la solución

En este apartado se detallará el trabajo realizado, explicando tanto el proceso seguido para modelar cada sistema, así como explicando cada uno de los progresivamente más complejos sistemas modelizados, y la dificultad adicional que entraña cada uno para su modelado.

De esta forma, en esta sección se incluye un primer apartado principal comentando el entorno de desarrollo y funcionalidades empleadas, y luego, por cada sistema modelado, un apartado explicando en detalle el sistema en cuestión, y detallando el interés tras modelar cada uno.

6.1. Metodología y desarrollo

Para el desarrollo del proyecto, la herramienta empleada ha sido MATLAB, un entorno de cómputo numérico y programación ampliamente utilizado en ingeniería, tanto de forma profesional, como académica; en multitud de ámbitos, como análisis de datos, procesamiento de señales e imágenes, robótica... Uno de sus mayores usos es en el desarrollo de redes neuronales.

Se ha empleado MATLAB, en la versión R2022a, durante todo el proceso, desde el diseño, hasta la validación y análisis, pasando por el entrenamiento. Para esto, dentro de este entorno, se ha empleado la denominada *"Deep Learning Toolbox"*, que incorpora de forma general las funcionalidades correspondientes al diseño, creación, entrenamiento y validación de redes neuronales, sin necesidad de aplicar técnicas de *deep learning*, al contrario de lo que indica su nombre.

A continuación, se procede a detallar el proceso de diseño, entrenamiento y validación realizado, explicando en profundidad los pasos seguidos, y explicando las funciones empleadas para cada uno de los pasos.

6.1.1. Preparación de los datos de entrenamiento

Antes de proceder al entrenamiento de la red, se han obtenido los datos de las entradas de la red a emplear tanto en el entrenamiento, como en la validación. En el caso que nos ocupa, al tratarse de redes neuronales de tipo NARX, estas entradas se dividen en dos grupos: las entradas exógenas u (denominadas *inputs*), y salidas reales del sistema y (*targets*).

Para la obtención de estos datos, se deben obtener las variables de entrada y salida para un determinado número de muestras. En el caso de sistemas reales, estos datos se obtendrían mediante mediciones de las variables de entrada y salida en cuestión.

Sin embargo, en el caso que nos ocupa, los sistemas a modelar son sistemas simulados, por lo que la obtención de los datos de salida se realizará mediante la simulación de los sistemas para unas entradas definidas.

Para la generación de estas entradas, se ha desarrollado una función a la que se le ha denominado *camino aleatorio saturado* (*saturated random walk*).

Esta función tiene tres entradas: tamaño del vector, rango de salidas, y probabilidad de cambio p. En cuanto al funcionamiento, es similar a un *random walk* estándar: inicializa el primer elemento del vector a un valor aleatorio (distribución uniforme) dentro del rango, y tras esta primera muestra, para cada muestra subsecuente k, hasta rellenar el vector:

- Toma un nuevo valor aleatorio con probabilidad p, que es uno de los parámetros de entrada de la función
- En caso contrario, mantiene el mismo valor que la muestra anterior $k-1$

Sin embargo, la diferencia fundamental que se incorpora en este caso, es que el rango de valores aleatorios se aumenta en un porcentaje, a lo que se le denomina rango aumentado. Sin embargo, los valores dentro de este nuevo rango se saturan posteriormente, es decir, en caso de tomar valores fuera del rango original, estos se sustituyen por el límite inferior (si el valor es inferior a este) o el límite superior (si el valor es superior a este).

En el programa utilizado el incremento del rango es del 20 %, en cada dirección, es decir, si el rango original es $[-1,1]$, el rango total es de 2, por lo que el rango aumentado es $[-1{,}4,1{,}4]$. Así, al generar los valores aleatorios, en caso de que estuvieran en el rango $[-1{,}4,-1)$, estos se sustituyen por el límite inferior -1, y en caso de obtener valores en el rango $(1,1{,}4]$, estos se sustituyen por el límite superior 1.

El objetivo de esta modificación es generar una mayor cantidad de muestras en los extremos de los rangos de entrenamiento, donde se busca que la red neuronal aprenda bien estos valores, ya que, como se ha dicho, las redes MLP poseen poca capacidad de extrapolación.

Mediante el método del *camino aleatorio saturado*, se obtiene una señal de entrada con los beneficios del *camino aleatorio*, es decir, variedad de puntos de operación y cambios de entradas, con el beneficio adicional de un mejor entrenamiento de la red para los puntos en los límites del rango de aprendizaje.

La probabilidad de que, al generar un nuevo número aleatorio, este sea el límite superior o inferior (p_{sup}, p_{inf}) depende de los incrementos de rango realizados ($\Delta r_{sup}, \Delta r_{inf}$):

$$p_{sup} = \frac{\Delta r_{sup}}{1 + \Delta r_{sup} + \Delta r_{inf}} \qquad p_{inf} = \frac{\Delta r_{inf}}{1 + \Delta r_{sup} + \Delta r_{inf}}$$

En el caso empleado, donde ambos incrementos son del 20 % ($\Delta r_{sup} = \Delta r_{inf} = 0{,}2$), esta probabilidad es de cerca del 15 % ($\frac{1}{7}$, para ser más exactos). Comparando la señal obtenida para ambos casos, se puede observar esta tendencia a obtener muestras en los extremos (Figura 12, misma semilla de generación, 100 muestras, y probabilidad de cambio $p = 0{,}4$).

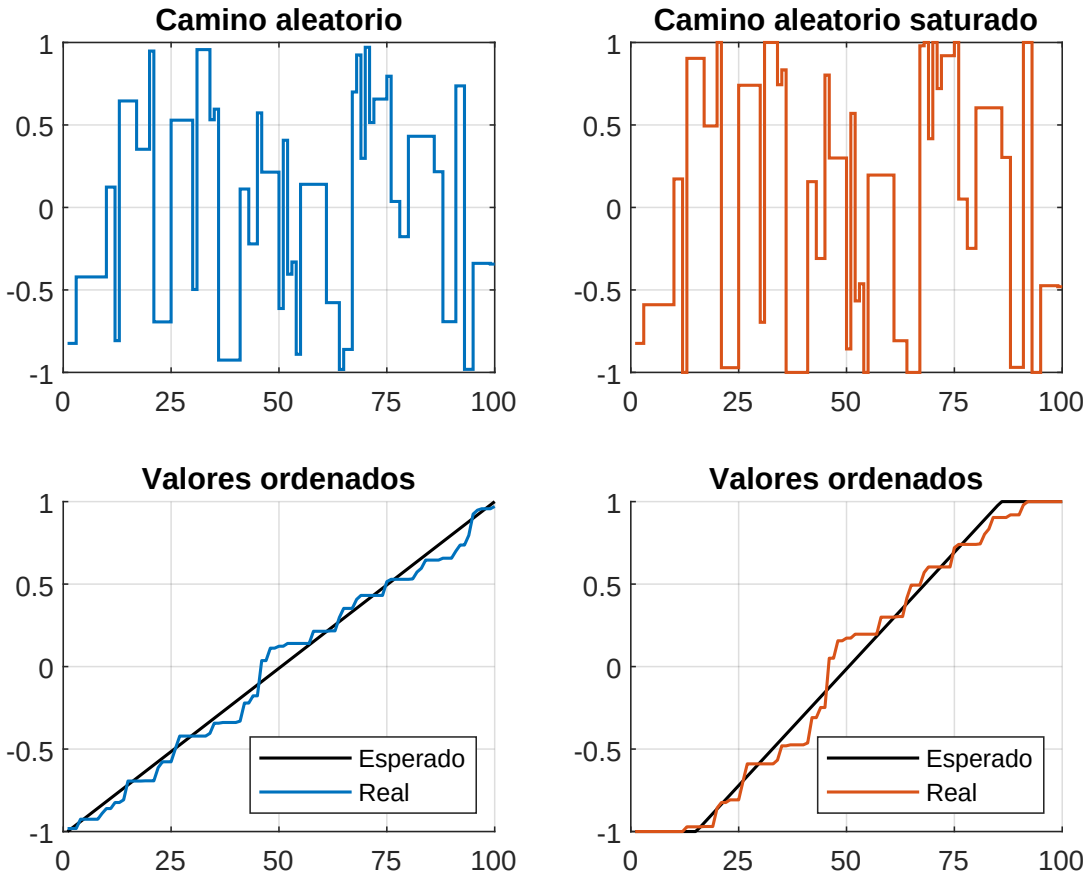

Figura 12: Comparación entre *camino aleatorio* y *camino aleatorio saturado*

6.1.2. Diseño de la red

Una vez obtenidos los datos de entrada, y de aplicar el sistema simulado a estos para obtener los datos de salida, se puede proceder a realizar el entrenamiento de la red neuronal que se empleará como modelo del sistema.

Además de estos datos, es necesario diseñar la red. La elección de la estructura de la red es el primer paso del diseño, en este caso, como se ha comentado, se han elegido redes neuronales con la estructura NARX en lazo abierto (Figura 13).

A continuación, se debe concretar la arquitectura de la red: se deben seleccionar las entradas de la red, y el número de capas ocultas y neuronas en cada capa oculta, aunque es conveniente empezar con una única capa oculta, por simplicidad, y por el Teorema de Aproximación Universal, que asegura que no es necesario emplear más de una.

Respecto al número de entradas y su retardo, y el número de neuronas, se realiza una primera estimación de estas basada en experiencia previa y conocimiento del sistema. Cabe destacar, que es mejor realizar una estimación por debajo del ideal para evitar sobreparametrizar la red neuronal, que aumentaría el coste computacional, y además reduciría la capacidad de generalización de la red.

Por ejemplo, si sabemos que por la dinámica del sistema, y según el tiempo de muestreo y pruebas realizadas, únicamente tienen un impacto notable en la salida, la entrada u actual y la anterior ($u(k)$ y $u(k-1)$), los entradas de la red no incluirán muestras anteriores de esa misma entrada ($u(k-2), u(k-3), \ldots$).

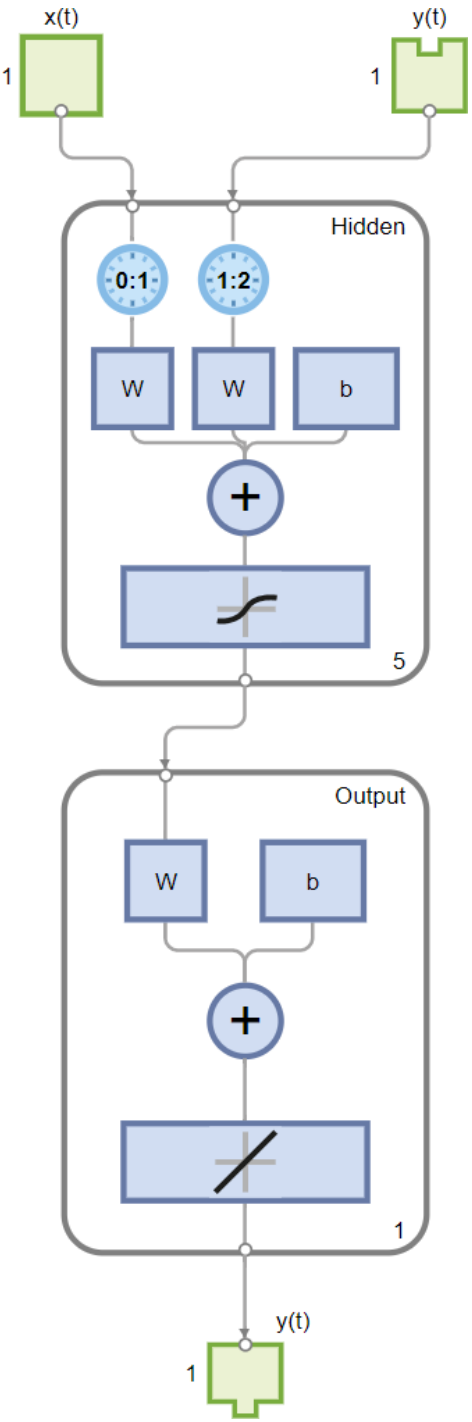

Figura 13: Visualización de una red neuronal NARX en MATLAB

6.1.3. Creación y entrenamiento de la red

Tras diseñar la red, estableciendo la estructura y arquitectura de la misma, puede dar comienzo el entrenamiento de la red neuronal.

Para realizar este proceso, como se ha comentado, se han empleado las herramientas de la *"Deep Learning Toolbox"* de MATLAB. En primer lugar, se ha creado una red neuronal NARX mediante la función `narxnet`. En esta función también se deben especificar las entradas, definiéndolas como los retardos de las dos entradas a emplear (*input* y *target*), los números de neuronas de cada capa oculta, la realimentación de la red, y el algoritmo de entrenamiento a emplear.

En el caso que nos ocupa, los dos últimos parámetros serán los mismos para todos los casos, ya que las redes empleadas han sido redes neuronales NARX en lazo abierto (sin realimentación de las salidas), y el algoritmo de entrenamiento empleado ha sido el algoritmo de Levenberg-Marquardt. Respecto a los retardos de las entradas y el número de neuronas, estos variarán para cada aplicación. Aún así, al solo utilizar redes con una única capa oculta, se debe especificar un único número de neuronas.

Así, en un caso donde se desee crear una red `net` con 8 neuronas en la capa oculta, y cuyas entradas sean $[u(k), u(k-1), y(k-1)]$, la instrucción a utilizar sería:

```
net = narxnet(0:1,1,8,'open','trainlm');
```

Una vez creada la red neuronal, para entrenarla se emplea la funcionalidad `train`. Sin embargo, se deben realizar varios pasos antes de poder realizar el entrenamiento, ya que el formato de los datos obtenidos y el formato empleado en las redes neuronales por MATLAB son diferentes.

Los datos de entrenamiento se obtienen como vectores o matrices, donde cada fila corresponde a una entrada del sistema y cada columna a un instante de tiempo o muestra. Por ejemplo, en un sistema con 2 entradas, si tomamos 1000 muestras, los datos tendrán el formato de una matriz de tamaño 2x1000.

Sin embargo, MATLAB emplea un formato diferente para las redes neuronales que, en vez de una matriz numérica, emplea el denominado formato *celda* (*cell*). En este formato, los datos son una fila de celdas, donde cada celda i almacena un vector con todas las entradas de la muestra i.

En adición a esta diferencia entre los tipos de variable de los datos, se añade que MATLAB divide los datos, diferenciando los denominados *estados iniciales* del resto de datos. Estos estados iniciales son aquellas muestras que, debido a los retardos de la red, no pueden utilizarse como "muestra actual"para el cálculo.

Para explicar esto, es mejor suponer un ejemplo. Suponiendo que la muestra más antigua que usa nuestra red es $u(k-3)$, eso significa que para las tres primeras muestras ($k = 1, 2, 3$), la entrada correspondiente a dicha muestra más antigua no existiría. MATLAB soluciona esto empezando desde los cálculos desde la cuarta muestra, y definiendo las tres primeras muestras como *estados iniciales*.

Sabiendo cómo opera MATLAB y que formatos son los que usa, se puede hacer manualmente, cambiando de tipo los datos y dividiéndolos según MATLAB establece. Sin embargo, el propio entorno ya incluye estas funcionalidades, en las funciones `tonndata` y `preparets`.

La primera función tiene tres parámetros, y son sencillos: el primero son los datos a transformar, el segundo indica si cada muestra (instante de tiempo) se almacena en una columna (en este caso es `true`), y el último indica si los datos son una *celda* (en este caso es `false`, porque es una matriz o un vector).

Así, este cambio de tipo de variable de los datos de entrenamiento `inVector` y `tgVector` puede realizarse mediante el siguiente código:

```
inCell = tonndata(inVector,true,false);
tgCell = tonndata(tgVector,true,false);
```

La segunda función es más compleja, ya que posee cuatro parámetros de entrada, y otros cuatro de salida. Los parámetros de entrada son, en orden: red neuronal, entradas (obtenidas de la anterior función), *targets* no realimentados (vacío en este caso), y *targets* realimentados (estos son los *targets* en el caso de las redes NARX).

Por su lado, las cuatro salidas son los datos en el formato que MATLAB emplea: datos de entrada (incluye *inputs* y *targets*), estados iniciales de las entradas, estados iniciales de las capas, y datos de *targets* (para las salidas de la red).

```
[inData,inStat,layStat,tgData] = preparets(inCell,{},tgCell);
```

Finalmente, es necesario modificar diversos parámetros de la red. Estos parámetros son los ratios de división de muestras `divideParam`. Estos indican que fracción de las muestras se emplean en el entrenamiento (`trainRatio`), en la validación (`valRatio`), y en la comprobación (`testRatio`) de la red.

En este caso, al hacer la validación de forma independiente al entrenamiento, se desea que todos los datos se usen para el entrenamiento.

```
net.divideParam.trainRatio = 1;
net.divideParam.valRatio = 0;
net.divideParam.testRatio = 0;
```

De esta forma, una vez realizadas todas estas operaciones, tanto la red como los datos de entrenamiento están configurados y con el formato adecuado para poder realizar el entrenamiento mediante la instrucción `train`:

```
[net, trRec] = train(net,inData,tgData,inStat,layStat);
```

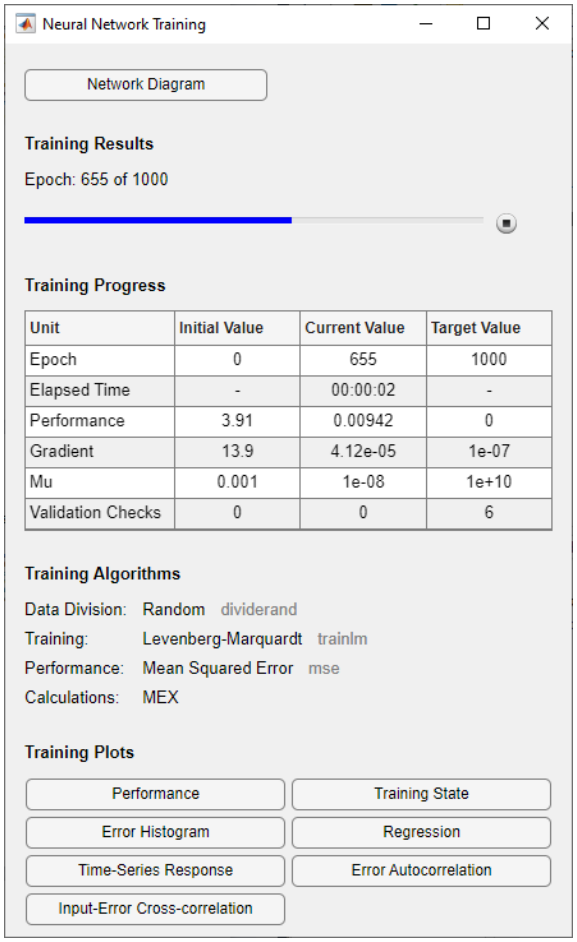

Figura 14: Ventana de entrenamiento de red

Además de la red entrenada, esta función también crea un registro del entrenamiento (`trRec`), donde almacena diversos datos sobre los parámetros y resultados del entrenamiento.

Durante el entrenamiento, se crea una ventana (Figura 14) con diversa información relevante. En ella aparecen los valores mínimo, actual y máximo de diversas variables, entre otras:

- Iteración actual (*epoch*)
- Rendimiento de la red (*performance*) calculada según el algoritmo que aparece en la parte inferior, en el caso que nos ocupa, error cuadrático medio (MSE, *Mean Squared Error*)
- Gradiente del error (*gradient*)

Estos parámetros de entrenamiento (límites inferior y superior), están almacenados dentro de la variable `net`, donde son denominados `trainParam` y pueden modificarse de forma similar a los ratios de división de muestras mencionados anteriormente.

Además de observar la evolución de estos datos durante el entrenamiento, desde la parte superior de esta ventana también se puede generar un diagrama de la red, similar al de la Figura 13, o generar, desde la parte inferior, gráficos en tiempo real de la evolución de diversas variables de entrenamiento como el rendimiento.

6.1.4. Validación de la red

Una vez finalizado el entrenamiento de la red neuronal, el último paso antes de tomar el modelo neuronal como bueno es realizar la validación del mismo.

Este paso consiste, como se ha explicado, en comprobar que la red neuronal representa correctamente el sistema modelizado y es capaz de generalizar el comportamiento de este más allá de los datos de entrenamiento, cualquier entrada dentro del rango de entrenamiento.

Para esto, a diferencia del diseño, creación y entrenamiento de la red, no se han empleado las funciones propias de MATLAB, sino que se ha desarrollado de cero una función equivalente que realice las operaciones de la red neuronal.

El objetivo de esto es reducir el coste computacional de la red, ya que las redes de MATLAB (formato `network`) almacenan muchísima información sobre la red, como datos de parámetros de entrenamiento, detalles sobre mapeo de entradas, o parámetros de inicialización de los valores de la red, entre otros muchos. Esto resulta muy útil, pero muy pesadas a la hora del cálculo, especialmente con vistas a su implantación como modelos para control en tiempo real.

6.1.4.1. Datos de la red reducida y cálculo matricial de salidas

De esta forma, se ha optado por una función que extrae los parámetros relevantes para el cálculo de las salidas de la red (pesos y *bias* de las neuronas, retardos y mapeo de entradas,...), obteniendo así una estructura (formato `struct`) con esos datos, a la que se ha denominado *red reducida*.

Es esta red reducida la que se ha empleado para el cálculo de las salidas de la red. Además, debido a cómo guarda MATLAB la información de la red, podemos usar la gran capacidad de MATLAB para el cálculo matricial para acelerar los cálculos.

En primer lugar, las entradas a la red se dividen en dos vectores, *inputs* y *targets*. Por consistencia con los formatos que emplea MATLAB, estos vectores son de tamaño $p \times 1$ para el vector de *inputs*, y de tamaño $q \times 1$ para el de *targets*.

El formato empleado puede visualizarse con facilidad con un ejemplo para el caso de una red con 3 entradas exógenas y 2 salidas, donde las entradas de la red son $[u(k), u(k-1), y(k-1), y(k-2)]$. En este ejemplo, los vectores de entrada serían:

$$U_{in} = \begin{bmatrix} u_1(k) \\ u_2(k) \\ u_3(k) \\ u_1(k-1) \\ u_2(k-1) \\ u_3(k-1) \end{bmatrix} \qquad U_{tg} = \begin{bmatrix} y_1(k-1) \\ y_2(k-1) \\ y_1(k-2) \\ y_2(k-2) \end{bmatrix}$$

De esta forma, p es el producto entre el número de entradas exógenas (número de entradas del sistema) y el número de muestras distintas de esta entrada (2, ya que usa las muestras k y $k-1$); y q es el producto entre el número de *targets* (igual al número de salidas del sistema) y el número de muestras distintas de esta entrada (muestras $k-1$ y $k-2$).

Estas entradas entran a la red y pasan por la capa de entrada. El objetivo de esta capa es escalar y normalizar las entradas de una forma lineal ($y = mx + n$), de forma que el rango de datos empleado en el entrenamiento se mapee al rango $[-1, 1]$.

Para realizar esto, se almacenan en la red reducida un punto de la recta y la pendiente m. El punto corresponde al punto del valor mínimo del rango de entrenamiento que, como se ha comentado, se mapea al -1, por lo que el punto es $(x_{min}, -1)$. Como es de esperar, las transformaciones de las entradas de *inputs* y *targets* son diferentes, ya que trabajan en rangos distintos.

Con ambos datos, se puede escalar cualquier punto de las entradas, y si están dentro del rango de entrenamiento (que así debería ser para evitar extrapolaciones y mejorar el rendimiento de la red), se mapeará a un valor en el rango $[-1, 1]$:

$$\begin{cases} \mathbf{u_{in}} = -\mathbf{1} + \mathbf{m_{in}} \cdot (\mathbf{U_{in}} - \mathbf{x_{min_{in}}}) \\ \mathbf{u_{tg}} = -\mathbf{1} + \mathbf{m_{tg}} \cdot (\mathbf{U_{tg}} - \mathbf{x_{min_{tg}}}) \end{cases}$$

Respecto a la capa oculta, MATLAB guarda los pesos de las neuronas en matrices, donde cada fila corresponde a cada una de las n neuronas de la capa, y cada columna a un elemento de la entrada. Como la capa oculta recibe dos entradas (*inputs* y *targets*), se tienen dos matrices de pesos, una por cada entrada. La matriz de pesos de *inputs* es de tamaño $n \times p$, y la matriz de pesos de *targets* es de tamaño $n \times q$.

Por otro lado, los *bias* de las neuronas se almacenan en un vector de tamaño $n \times 1$, donde cada elemento corresponde a cada una de las neuronas, de forma similar similar a las matrices de pesos. De esta forma, los parámetros de las neuronas de la capa oculta se almacenan en tres matrices:

$$W_{in} = \begin{bmatrix} w_{1_{in1}} & w_{1_{in2}} & \cdots & w_{1_{inp}} \\ w_{2_{in1}} & w_{2_{in2}} & \cdots & w_{2_{inp}} \\ w_{3_{in1}} & w_{3_{in2}} & \cdots & w_{3_{inp}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_{in1}} & w_{n_{in2}} & \cdots & w_{n_{inp}} \end{bmatrix} \quad W_{tg} = \begin{bmatrix} w_{1_{tg1}} & w_{1_{tg2}} & \cdots & w_{1_{tgq}} \\ w_{2_{tg1}} & w_{2_{tg2}} & \cdots & w_{2_{tgq}} \\ w_{3_{in1}} & w_{3_{in2}} & \cdots & w_{3_{inp}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_{tg1}} & w_{n_{tg2}} & \cdots & w_{n_{tgq}} \end{bmatrix} \quad B_h = \begin{bmatrix} b_{h1} \\ b_{h2} \\ b_{h3} \\ \vdots \\ b_{hn} \end{bmatrix}$$

Así, con estas matrices y vectores, se puede operar para obtener la salida de las neuronas de capa oculta, recordando que se aplica la función de activación de la neurona, en este caso, la función tangente hiperbólica (tanh(x)), denominada en MATLAB tangente sigmoidal (`tansig`). De esta forma, la salida de las neuronas de la capa oculta (o_h) es un vector de tamaño $n \times 1$, siendo cada elemento la salida de cada neurona:

$$\mathbf{o_h} = \tanh(\mathbf{W_{in}} \cdot \mathbf{u_{in}} + \mathbf{W_{tg}} \cdot \mathbf{u_{tg}} + \mathbf{B_h}) \qquad o_h = \begin{bmatrix} o_{h1} \\ o_{h2} \\ o_{h3} \\ \vdots \\ o_{hn} \end{bmatrix}$$

La capa de salida funciona de forma similar a la capa oculta, sin embargo, esta capa solo tiene una entrada (las salidas de la capa oculta), por lo que tendrá una única matriz de pesos, de tamaño $o \times n$, siendo o el número de salidas de la red neuronal; además de su vector de *bias* de tamaño $o \times 1$:

$$W_o = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{o1} & w_{o2} & \cdots & w_{on} \end{bmatrix} \qquad B_o = \begin{bmatrix} b_{o1} \\ b_{o2} \\ \vdots \\ b_{on} \end{bmatrix}$$

De forma similar a la capa oculta, operando estas matrices y vectores, se obtiene la salida final de la red. En este caso, la función de activación es lineal, por lo que la salida de la red (o_{net}) es:

$$\mathbf{o_{net}} = \mathbf{W_o} \cdot \mathbf{o_h} + \mathbf{B_o} \qquad o_{net} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_o \end{bmatrix}$$

Cabe destacar que esta salida (o_{net}) debe desnormalizarse, siguiendo el proceso inverso al realizado para la entrada del *target* (ya que usa los mismos parámetros). De esta forma, teniendo los valores x_{min} y m correspondientes al *target*, podemos obtener la salida real (O_{net}) mediante la siguiente operación:

$$x_{min_{out}} = x_{min_{tg}} \qquad m_{out} = m_{tg}$$
$$\mathbf{O_{net}} = \mathbf{x_{min_{out}}} + \frac{\mathbf{o_{net}} - (-\mathbf{1})}{\mathbf{m_{out}}}$$

De esta forma, en resumen, la red reducida debe contener las matrices de pesos y vectores de *bias*, tanto de la capa oculta, como de la capa de salida, además de los parámetros de normalización (x_{min}, y pendiente m) de las dos entradas. Así, la función de reducción de la red desarrollada toma una red de tipo `network`, y crea una red reducida de tipo `struct`, con estos datos.

Por otro lado, se ha desarrollado la función que reproduce las operaciones de la red aplicando las operaciones matriciales mencionadas, capa a capa: capa de entrada (normalización de entradas), capa oculta, y capa de salida (cálculo de salida y desnormalización). A esta función se la denominará, por simplicidad, *función NARX*.

6.1.4.2. Validación de la red en lazo abierto

Tal y como se ha comentado, el método más común para validar una red neuronal es pasar por la red un conjunto de datos de entradas (*inputs* y *targets*) distinto al empleado en el entrenamiento, y calcular el rendimiento de la red frente a este nuevo conjunto de datos. De esta forma, se comprueba que la red reproduzca el comportamiento del sistema, y sea capaz de generalizar dicho comportamiento para entradas distintas de las que ha sido entrenado.

Para realizar este procedimiento, se genera un nuevo conjunto de datos de entrada dentro del rango de valores donde ha sido entrenada la red, y sus respectivas salidas. Para esto se ha empleado el mismo método que para la generación de datos de entrenamiento, la función *camino aleatorio saturado*.

Pasando las entradas por el sistema a modelizar, se obtienen las salidas, de forma equivalente a los datos de entrenamiento. Es tras esto donde se diferencian los dos procesos.

Para la validación, estos datos de entrada se insertan, muestra a muestra, en la función NARX. Tras pasar todas las muestras, se compara la salida obtenida con la salida obtenida teóricamente, y se obtiene el rendimiento de validación. En este caso, a diferencia del entrenamiento, para el rendimiento de la validación, se calculan tanto el *error cuadrático medio* (MSE, *Mean Square Error*), como el *error absoluto máximo* (MaAE, *Maximum Absolute Error*).

Ambas son relevantes, la primera porque da una idea general del grado de corrección de la salida obtenida, y la segunda porque advierte del orden de magnitud de los errores de modelado cometidos, muy relevante a la hora de diseñar el sistema de control, ya que grandes errores pueden causar acciones de control excesivas que pueden empeorar el control, y en el peor de los casos, inestabilizar el sistema, si no se han tenido en cuenta estos casos.

6.1.4.3. Validación de la red en lazo cerrado

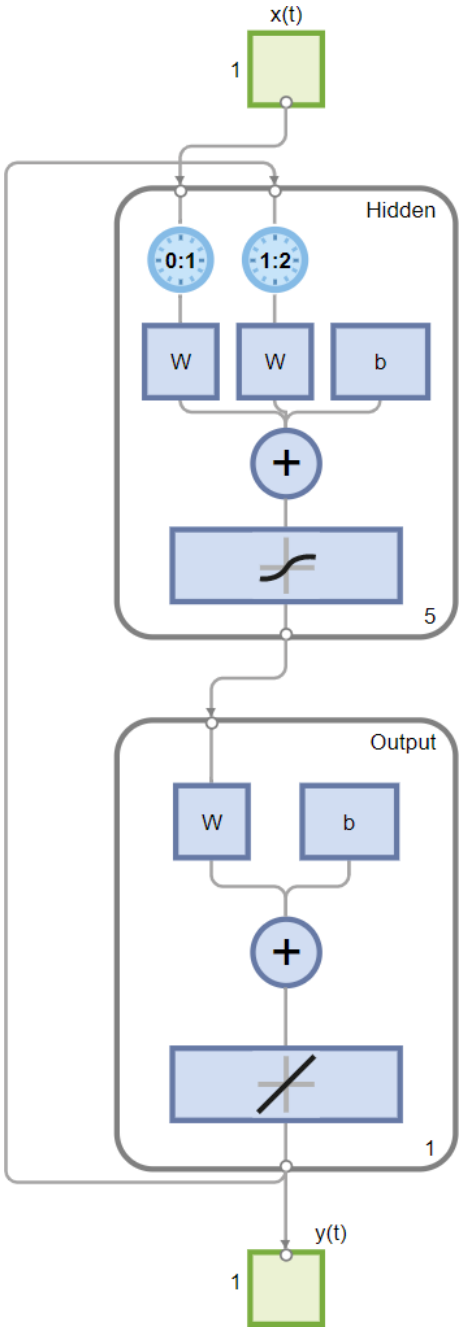

Figura 15: Red neuronal NARX en lazo cerrado en MATLAB

En adición a la anterior, también se ha realizado, debido a que se busca que estos modelos sean empleados en sistemas de control predictivo, una validación adicional de la red, esta vez, en lazo cerrado.

Esta validación es especialmente relevante, ya que un mal comportamiento en lazo cerrado de la red puede causar que las predicciones de las salidas dentro del horizonte de control del controlador sean equivocadas, y por tanto, el control realizado no sea adecuado.

Para conseguir esto, se realimentan las predicciones $\hat{y}(k)$ de la propia red, sustituyendo las salidas reales $y(k)$ (*target*). Así, en la red en lazo cerrado (Figura 15), es necesario insertar a la red únicamente las entradas exógenas $u(k)$ (*input*).

De esta forma, se revisa si la red es suficientemente robusta como para poder corregir el efecto de la diferencia en la entrada, entre las predicciones realimentadas, que no son exactas, y las salidas reales.

En el caso en que la red no sea capaz de corregir las diferencias entre estos valores, los errores pueden acumularse, causando una divergencia entre la salida calculada y la real, que crece por cada muestra en que la red trabaja en lazo cerrado.

6.1.5. Integración del desarrollo: barrido de parámetros

Hasta el momento, se han explicado los distintos procesos involucrados en el desarrollo de un modelo neuronal, desde la preparación de los datos de entrenamiento, a la validación.

Sin embargo, aunque un conocimiento del problema y sistema a modelar, y la experiencia previa en el ámbito juegan un papel fundamental en concretar la arquitectura de la red (entradas de la red, número de neuronas en la capa oculta...), encontrar el modelo neuronal óptimo, o al menos uno bueno, suele tratarse de un proceso iterativo, en el que incrementalmente se acerca a la solución final.

Esto, unido a la aleatoriedad propia del entrenamiento de la red neuronal, debida a la inicialización aleatoria de los parámetros de la red neuronal (pesos y *bias*) al comienzo del entrenamiento, debido a que el algoritmo de entrenamiento no garantiza que el entrenamiento minimice el error de forma global; hace que un proceso basado en diseñar, crear, entrenar, y validar modelos neuronales de forma individual sea terriblemente ineficiente.

Es este problema el que soluciona el código final desarrollado en MATLAB, que integra todos estos procesos en un solo código principal, facilitando el desarrollo de modelos neuronales. Adicionalmente, dentro del código los distintos procesos están suficientemente definidos, separados, y modularizados, de forma que se simplifique el código, y sea sencillo generalizarlo, ayudando a optimizar el desarrollo de futuros modelos neuronales, uno de los objetivos del trabajo.

Para evitar los dos problemas del desarrollo de modelos neuronales, se emplea el denominado *barrido de parámetros con repetición*. Mediante este método, el programa realiza el proceso completo de entrenamiento y validación de la red para cada una de las combinaciones de parámetros dentro de un rango prefijado de forma inicial, permitiendo comparar y visualizar el rendimiento de cada una de las arquitecturas de red. Por otro lado, para mitigar la aleatoriedad del entrenamiento, para cada combinación de parámetros, se entrenan y validan múltiples redes, también especificado de forma inicial.

Este proceso es simple, lo que simplifica el código, y aunque puede parecer poco eficiente, debido a que el aumento de los rangos de parámetros aumenta el número de redes a entrenar en gran medida, para el desarrollo de este trabajo, debido a la complejidad de los sistemas tratados, el conocimiento de los sistemas, y experiencia, el número de redes a entrenar entra dentro de un número razonable. Por otro lado, poseer datos de rendimiento de ese número de redes también permite comprobar la sensibilidad del rendimiento de la red respecto a esos parámetros.

Respecto al propio programa desarrollado, en primer lugar se especifican los rangos de parámetros a barrer (número de neuronas de la capa oculta, retardos de las muestras de *input*, y retardos de las muestras de *target*), el número de repeticiones de cada red, y los conjuntos de datos de entrenamiento y validación (entrada del sistema y salida del sistema/*target*).

Para almacenar todas las redes neuronales, se crea una variable llamada `netData`, donde se almacenan todas las redes, y sus rendimientos de entrenamiento y validación. Esta variable es una celda bidimensional, donde cada fila corresponde a un número de neuronas, y cada columna a una distribución de entradas (combinación de retardos de *input* y *target*) (Figura 16).

Por tanto, en cada elemento de la celda, la arquitectura de la red es idéntica. Así, en cada uno de ellos, se almacenan todas el conjunto de todas las repeticiones de dicha red (cada una de las barras de la Figura 16), con sus diferentes rendimientos, debido a la aleatoriedad del entrenamiento.

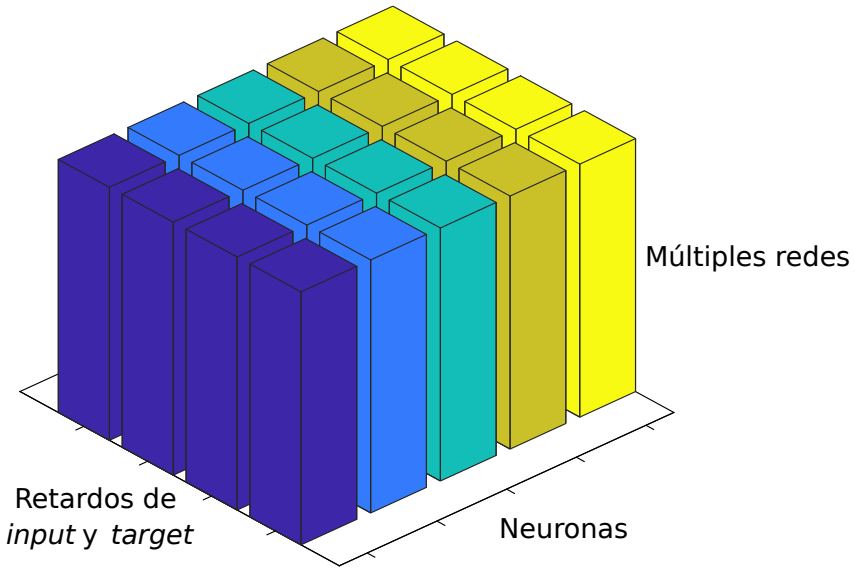

Figura 16: Visualización de la estructura de datos *netData*

Además, para mejorar la legibilidad de la estructura de datos al usuario, se han añadido filas y columnas adicionales con texto, que sirven como cabeceras de tanto las filas, como las columnas.

Para realizar el entrenamiento y validación de las redes, se ha empleado un bucle doble para rellenar la celda bidimensional elemento a elemento, un bucle recorriendo las filas, y otro cada elemento de estas.

Por cada elemento, otro bucle repite el número de veces necesario el entrenamiento y las dos validaciones, estando cada proceso en subfunciones independientes, para facilitar la modularización del programa.

Una vez entrenadas y validadas todas las redes, es necesario procesar esta estructura de datos para obtener información relevante. Para este fin, se ha desarrollado otra función que toma esta estructura de datos y, por cada elemento (arquitectura de red) toma la media aritmética de los rendimientos de validación para todas las repeticiones.

De esta forma, se obtiene un único valor que permite comparar las distintas arquitecturas de red entre sí, para obtener la mejor de ellas. La función de procesamiento de datos también toma la *mejor* red entrenada entre todas las repeticiones.

Sin embargo, elegir la mejor red de todas las repeticiones puede resultar complejo: una de ellas puede dar mejores resultados en la validación en lazo abierto, mientras que otra es más estable en la validación en lazo cerrado.

Para estos casos, se ha decidido que el usuario determine el criterio, eligiendo cuál de los tres rendimientos se busca minimizar, ya sea MSE, MaAE, o divergencia en lazo cerrado (medida mediante MSE en lazo cerrado).

Sin embargo, la selección de la arquitectura final se ha dejado completamente en manos del usuario, sin ninguna función que lo haga. Para ayudar al usuario a elegir la mejor red de todas, que será el modelo neuronal obtenido finalmente, sí se ha desarrollado una función que grafica los rendimientos de validación para poder compararlos fácilmente.

Esta función, además de graficar los gráficos comparativos de rendimientos de validación, que también es útil para comprobar la sensibilidad de estos frente a los parámetros del barrido; también grafica, para todas las arquitecturas, la comparación entre las salidas reales $y(k)$, y las salidas predichas por la red $\hat{y}(k)$, permitiendo observar la localización y magnitud de los errores cometidos por las redes neuronales.

Adicionalmente, para poder observar todas estas imágenes en un futuro, especialmente teniendo en cuenta el volumen de imágenes que se genera,ya que se genera una por combinación de parámetros, además de las comparativas de rendimientos; estas se guardan como archivos en una carpeta especificada.

En el Anexo I se puede encontrar todo el código desarrollado y empleado para el desarrollo y generación de los modelos neuronales de este trabajo, separado en las distintas funciones y subfunciones creadas.

6.2. Sistemas objetivo a modelizar

Tal y como se ha explicado en la introducción de la sección, tras explicar la metodología seguida para el desarrollo de los modelos neuronales, se procede a realizar una introducción y descripción de los sistemas modelizados.

En primer lugar, cabe señalar el hecho de que se han empleado sistemas sintéticos, en vez de modelos reales. Esto es debido a que el objetivo del trabajo es desarrollar modelos neuronales de sistemas complejos, y demostrar su eficacia. Tanto la aplicación de estos en esquemas de control, como la modelización de sistemas reales, quedan fuera del alcance del presente trabajo.

Sin embargo, para el trabajo que nos ocupa, los modelos sintéticos permiten comprobar el rendimiento de los modelos neuronales frente a diversas fuentes de complejidad del sistema. Por esta razón, cada sistema modelado incluye una nueva fuente de complejidad en la modelización, realizando una progresión incremental de la complejidad.

Los sistemas modelizados ya han sido diseñados y empleados con anterioridad en otros trabajos del ámbito [13]. Estos sistemas se han denominado SNL5 y SNL1, y se definen con las siguientes ecuaciones (para abreviar las ecuaciones, se representa como $y(k) = y_k$):

- **SNL5:** Extraído de [14].

$$y_{k+1} = \frac{1{,}5 \cdot y_k \cdot y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + 0{,}7 \cdot \sin\left(0{,}5\left(y_k + y_{k-1}\right)\right) \cdot \cos\left(0{,}5\left(y_k + y_{k-1}\right)\right) + 1{,}2u_k \tag{6.1}$$

- **SNL1:** Extraído de [1].

$$y_{k+1} = \frac{y_k}{1 + y_k^2} + u_k^3 \tag{6.2}$$

Ambos sistemas muestran comportamientos no-lineales relevantes y de distintos tipos (Figura 17):

- En el sistema SNL5, los puntos de operación siguen una función con ondulaciones, con múltiples puntos con grandes no-linealidades.
- El sistema SNL1 también es no-lineal, pero, en este caso, la salida y es muy sensible a la entrada u, debido a que esta aparece elevada a la tercera potencia.

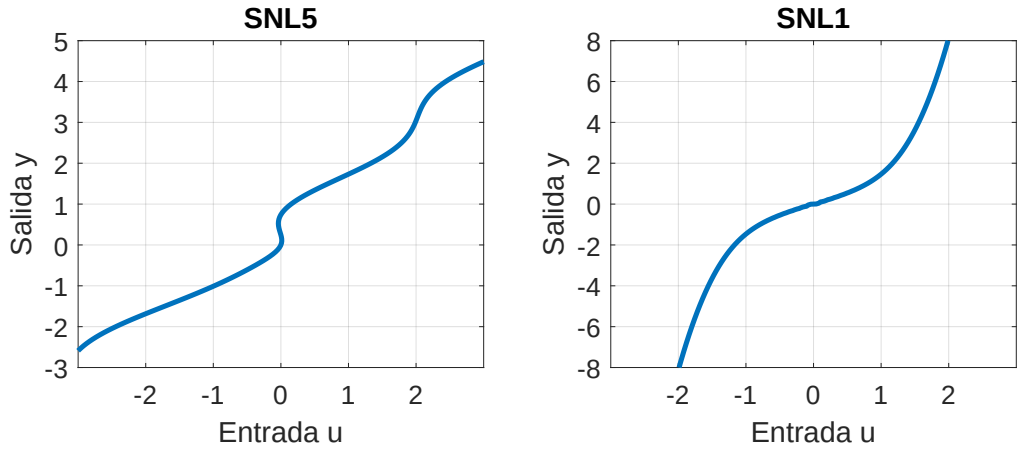

Figura 17: Entradas y salidas de los sistemas en régimen estacionario

6.2.1. Sistema SISO no-lineal

El primero de los sistemas modelizados es el sistema SNL5. Como se ha comentado, se ha comenzado por un sistema simple, sobre el que se basan el resto.

Este sistema es un sistema con una única entrada y una única salida (SISO), no-lineal, que permite realizar una primera aproximación al problema que nos ocupa.

De esta forma, el sistema a modelar es el dado en la Ecuación 6.1. Sin embargo, para el caso que nos ocupa, es mejor reescribirla de forma que la salida calculada sea la salida actual y_k.

Aunque este paso puede resultar redundante, es mejor escribirla así para ver, interpretar, y extraer información de ella con mayor facilidad ya que, en cuanto a los retardos empleados en MATLAB, la salida de la red se define como la salida de la muestra actual (retardo 0). Así, la ecuación que representa el sistema a modelar es:

$$y_k = \frac{1{,}5 \cdot y_{k-1} \cdot y_{k-2}}{1 + y_{k-1}^2 + y_{k-2}^2} + 0{,}7 \cdot \sin\left(0{,}5\left(y_{k-1} + y_{k-2}\right)\right) \cdot \cos\left(0{,}5\left(y_{k-1} + y_{k-2}\right)\right) + 1{,}2u_{k-1}$$

Esta ecuación puede ser simplificada aplicando identidades trigonométricas, resultando en la ecuación final del primer sistema modelado:

$$y_k = \frac{1{,}5 \cdot y_{k-1} \cdot y_{k-2}}{1 + y_{k-1}^2 + y_{k-2}^2} + 0{,}35 \cdot \sin\left(y_{k-1} + y_{k-2}\right) + 1{,}2u_{k-1} \tag{6.3}$$

De esta forma, tras reescribirla, en la Ecuación 6.3 puede observarse que las únicas entradas que tienen impacto en la salida y_k son $[u_{k-1}, y_{k-1}, y_{k-2}]$. De esta forma, gracias al conocimiento del sistema, se conocen los retardos máximos de las entradas, y pueden establecerse sus rangos de barrido sin necesidad de estimación.

En este caso, los retardos de las entradas exógenas (*input*) serán 0 o 0/1, mientras que los retardos de *target* serán 1 o 1/2.

Por otro lado, debido a que se trata se un sistema SISO, y no excesivamente complejo, el número de neuronas se establecerá en el rango de entre 2 y 6. Esta se trata de una estimación basada en la experiencia previa, por lo que en caso de dar mal resultado, puede cambiarse.

Respecto al número de repeticiones de cada red, se establecerá en 4, para reducir el número de redes a entrenar, ya que no se estima que se requiera acercarse en exceso al modelo óptimo, debido a la baja complejidad del sistema.

6.2.2. Sistema MIMO no-lineal

Para el segundo sistema modelizado, se ha creado un sistema MIMO. Este sistema está compuesto por los dos sistemas no-lineales no presentados, SNL5 y SNL1, de forma independiente, es decir, sin acoplamiento entre ambos.

El objetivo de este segundo sistema es, una vez estudiada la viabilidad de los modelos neuronales para la modelización del anterior sistema SISO no-lineal, buscar ampliarlo a sistemas MIMO, mucho más complejos y cercanos a los sistemas reales.

La razón por la que se han elegido estos dos sistemas sintéticos para formar el sistema MIMO es por sus distintos comportamientos no-lineales, comentados anteriormente: en SNL5 los puntos de operación siguen una función con ondulaciones, mientras que en SNL1, esta relación sigue una función cúbica, creando grandes variaciones en la salida con pequeñas variaciones en la entrada, si está alejada del cero.

Así, se desea estudiar el comportamiento de la red intentando representar simultáneamente estos dos comportamientos. Por otro lado, se ha decidido no incluir acoplamiento entre ambos sistemas para estudiar el efecto de la complejidad introducida por las múltiples entradas y salidas, antes de afrontar otras fuentes de complejidad.

De esta forma, el sistema desarrollado, como en el caso anterior, asignando la salida al instante actual, y aplicando la identidad trigonómetrica para la ecuación correspondiente al sistema SNL5, se define por las siguientes ecuaciones:

$$\begin{cases} y_{1,k} = \dfrac{1{,}5 \cdot y_{1,k-1} \cdot y_{1,k-2}}{1 + y_{1,k-1}^2 + y_{1,k-2}^2} + 0{,}35 \cdot \sin\left(y_{1,k-1} + y_{1,k-2}\right) + 1{,}2u_{1,k-1} \\ \\ y_{2,k} = \dfrac{y_{2,k-1}}{1 + y_{2,k-1}^2} + u_{2,k-1}^3 \end{cases} \tag{6.4}$$

Observando la Ecuación 6.4, puede verse que, al igual que para el anterior sistema, las únicas entradas que tienen impacto en la salida y_k son $[u_{k-1}, y_{k-1}, y_{k-2}]$. Por tanto, gracias al conocimiento del sistema (ya que se conocen las ecuaciones), se pueden definir de igual manera los rangos de barrido de las entradas de la red: los retardos de *input* serán 0 o 0/1, mientras que los retardos de *target* serán 1 o 1/2.

Sin embargo, al ser en este caso, un sistema más complejo que el anterior, el número de neuronas del rango será mayor. En este caso, se estima que el número de neuronas esté entre 8 y 20. Además, para reducir el número de redes a entrenar, se establecerán valores en saltos de 2 neuronas ($8, 10, 12, 14, 16, 18, 20$).

Finalmente, debido a que el sistema es más complejo, y por tanto, los entrenamientos pueden resultar, por aleatoriedad, en redes poco optimizadas (lejos del error mínimo global de entrenamiento). Para lograr con mayor seguridad redes más optimizadas, se ha elevado el número de repeticiones, respecto al caso anterior, a 10 repeticiones para cada combinación de parámetros.

7. Análisis de los resultados

En este apartado se analizarán los resultados obtenidos durante el trabajo realizado, presentando y describiendo el modelo neuronal obtenido para cada uno de los sistemas modelados, además de presentar datos que justifiquen las elecciones realizadas.

Así, se incluye un apartado por cada sistema modelizado, donde se presentarán los resultados obtenidos durante el entrenamiento y validación de las redes, y al final de cada uno de ellos, finalmente se concluirá cuál es el modelo elegido.

7.1. Sistema SISO no-lineal

Para realizar el proceso de entrenamiento de este sistema, se ha empleado una entrada de entrenamiento, como se ha comentado en la sección anterior, siguiendo la función *camino aleatorio saturado*.

La entrada (Figura 18) contiene 300 muestras (instantes de tiempo), con una probabilidad de cambio en cada muestra del 20 %. Respecto al rango de valores, se ha decidido emplear valores de u en el rango $[-2{,}5, 2{,}5]$, debido a que representa una diversidad de puntos de operación y zonas con no-linealidades.

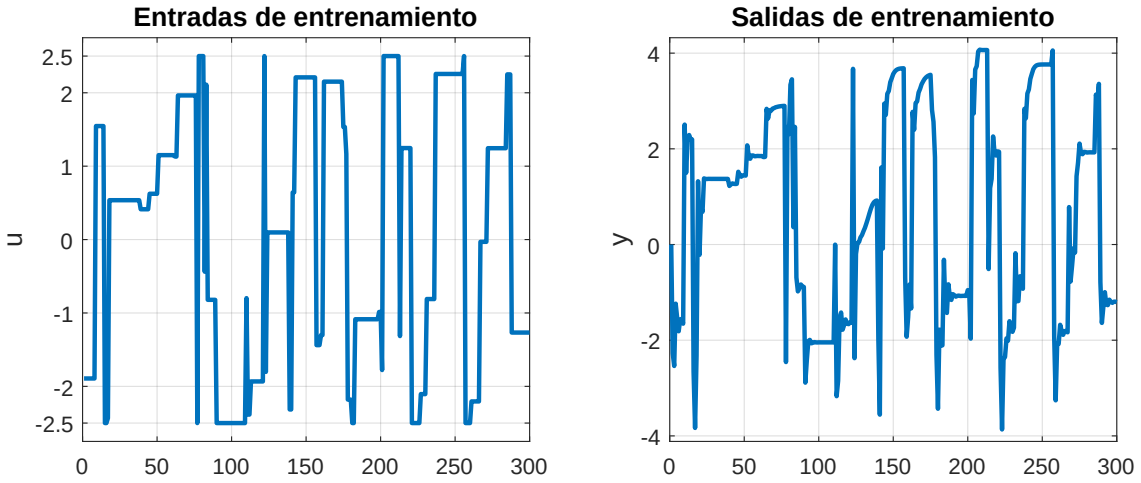

Figura 18: Datos de entrada y salida de entrenamiento para el sistema SISO no-lineal

Respecto a la entrada de validación, se ha empleado una entrada (Figura 19) con las mismas características que la entrada de entrenamiento.

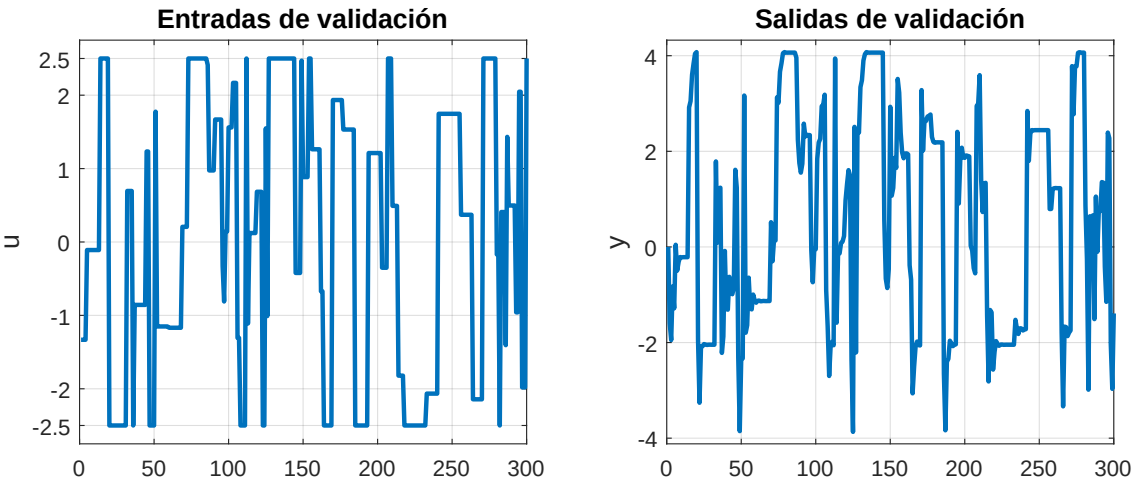

Figura 19: Datos de entrada y salida de validación para el sistema SISO no-lineal

Con estos datos de entrenamiento y validación, se ha procedido a la creación, entrenamiento y validación de las redes neuronales mediante el barrido de parámetros.

Tras realizar este proceso, y procesar la estructura de datos para obtener la mejor red para cada combinación de número de neuronas y retardos de *input* y *target*, se obtienen los siguientes gráficos comparativos de los tres rendimientos calculados: error cuadrático medio (MSE) y error absoluto máximo (MaAE) en lazo abierto (Figura 20), y MSE en lazo cerrado (Figura 21), como medida de la divergencia.

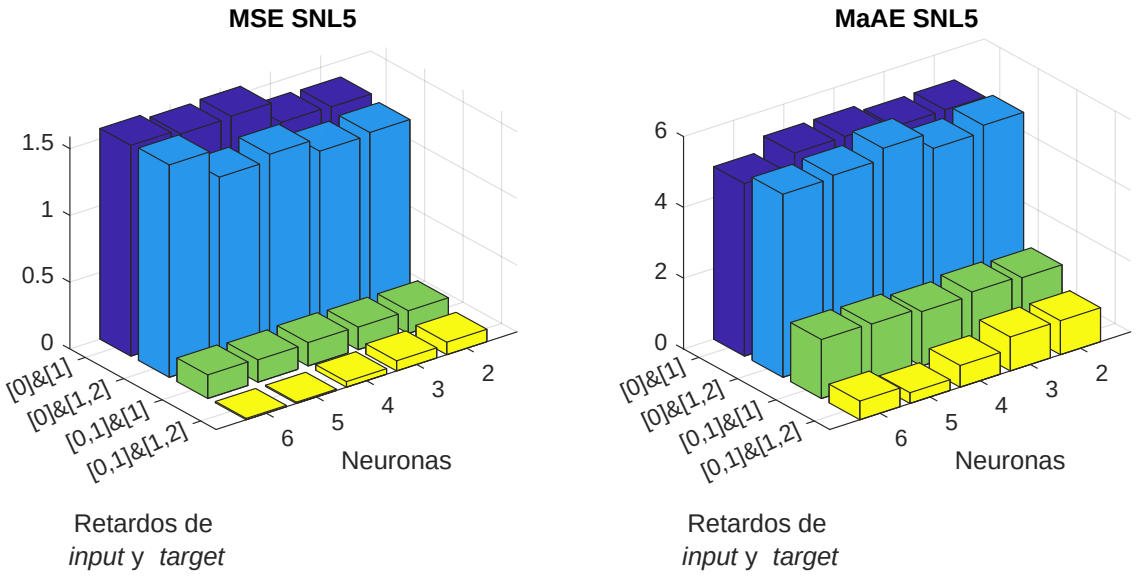

Figura 20: MSE y MaAE en lazo abierto de configuraciones de redes para el sistema SISO

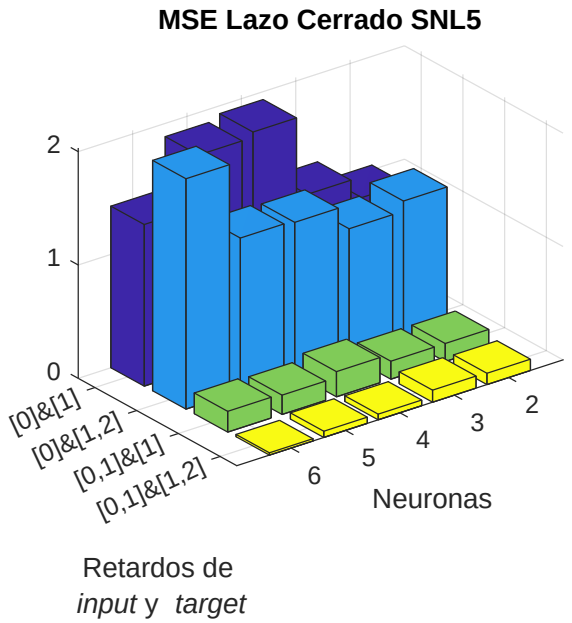

Figura 21: MSE en lazo cerrado de configuraciones de redes para el sistema SISO

Tal y como se observa en las Figuras 20 y 21, los parámetros barridos que tienen mayor impacto en la calidad del modelo neuronal obtenido son los retardos de entradas, particularmente los retardos de *input*, que reducen drásticamente el error en los tres casos graficados.

Por otro lado, insertar a la red un mayor número de muestras de *target* también mejora el rendimiento (menores índices de rendimiento, ya que se tratan de medidas de error: MSE, MaAE,...), aunque únicamente en el caso que los retardos de *input* sean 0 y 1, es decir, las entradas exógenas a la red sean $[u_k, u_{k-1}]$. Para estos casos, insertar y_{k-2} además de y_{k-1} genera una mejora adicional del rendimiento de los modelos neuronales.

Sin embargo, aunque sí se nota una mejoría al aumentar el número de neuronas, el impacto en el rendimiento de la red tampoco es muy elevado. De hecho, esta mejora es tan pequeña que, debido al aumento del tiempo de computación necesario para conseguirlo, es mejor elegir redes de tamaño medio.

Así, las redes que resultan más prometedoras son las redes con retardos de *input* 0 y 1, de 4 a 6 neuronas. A continuación, para poder comparar los datos de forma más exacta, se presentan las tablas que contienen los datos exactos para estas redes:

[inDel]&[tgDel]	4 neuronas	5 neuronas	6 neuronas
[0,1]&[1]	0.1751	0.1679	0.1767
[0,1]&[1,2]	0.0416	0.0096	0.0098

Tabla 4: MSE en lazo abierto de preselección de redes para el sistema SISO

[inDel]&[tgDel]	4 neuronas	5 neuronas	6 neuronas
[0,1]&[1]	1.5345	1.6432	1.6660
[0,1]&[1,2]	0.6128	0.3032	0.5264

Tabla 5: MaAE en lazo abierto de preselección de redes para el sistema SISO

[inDel]&[tgDel]	4 neuronas	5 neuronas	6 neuronas
[0,1]&[1]	0.2224	0.1705	0.1832
[0,1]&[1,2]	0.0500	0.0548	0.0202

Tabla 6: MSE en lazo cerrado de preselección de redes para el sistema SISO

Tras observar con más detalle los datos, se ha decidido descartar las redes con retardo de *target* 1, especialmente debido a que el error absoluto máximo que presentan (Tabla 5) es elevado. De las redes restantes, sólo falta elegir el número de neuronas en la capa oculta.

Así, se ha decidido emplear la red con 5 neuronas, ya que muestra un resultado sobresaliente, mejor incluso que la red con 6 neuronas excepto en la validación en lazo cerrado, aunque también obtiene muy buenos resultados. Respecto a la red con 4 neuronas, su rendimiento también es muy bueno, y tiene un coste computacional menor pero, comparativamente, su rendimiento en lazo abierto es peor que la red con 5 neuronas.

De esta forma, el modelo neuronal obtenido para representar el primer sistema objetivo es una red neuronal con 5 neuronas, retardos de *input* 0 y 1, y retardos de *target* 1 y 2, por lo que sus entradas son $[u_k, u_{k-1}, y_{k-1}, y_{k-2}]$.

Representando la respuesta del sistema durante la validación en lazo abierto (Figura 22) realizada, se puede comprobar que su respuesta es muy buena, siguiendo la salida real con error muy reducido, excepto en el límite superior del rango de valores, donde se aprecia un ligero error, aunque muy reducido.

Por otro lado, para esos mismos valores (límite superior del rango), la validación en lazo cerrado (Figura 23) presenta errores notables en los casos en los que la entrada se mantiene en el límite superior durante varias muestras. Sin embargo, también se comprueba que para el resto de valores, la respuesta es excelente, muy cercana a la salida real. De hecho, incluso en los momentos que la red toma como entrada predicciones anteriores con error (ya que trabaja en lazo cerrado), la respuesta sigue siendo buena, por lo que es capaz de rechazar estos errores.

Cabe destacar que, aunque no se ha representado, se ha comprobado que la red con 4 neuronas no posee este error en la validación en lazo cerrado, aunque presenta errores ligeramente mayores de forma general. De esta forma, esta red puede ser una alternativa en casos en los que este comportamiento en lazo cerrado del modelo seleccionado pueda resultar problemático.

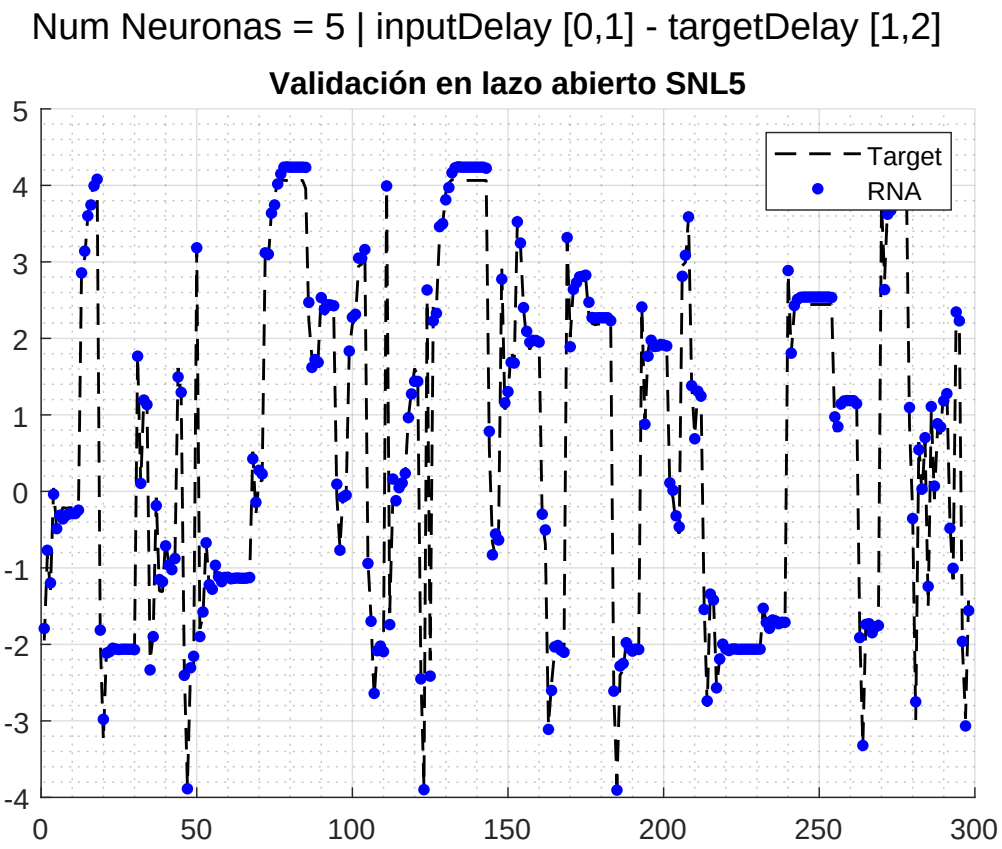

Figura 22: Respuesta del modelo neuronal para el sistema SISO obtenida para validación en lazo abierto

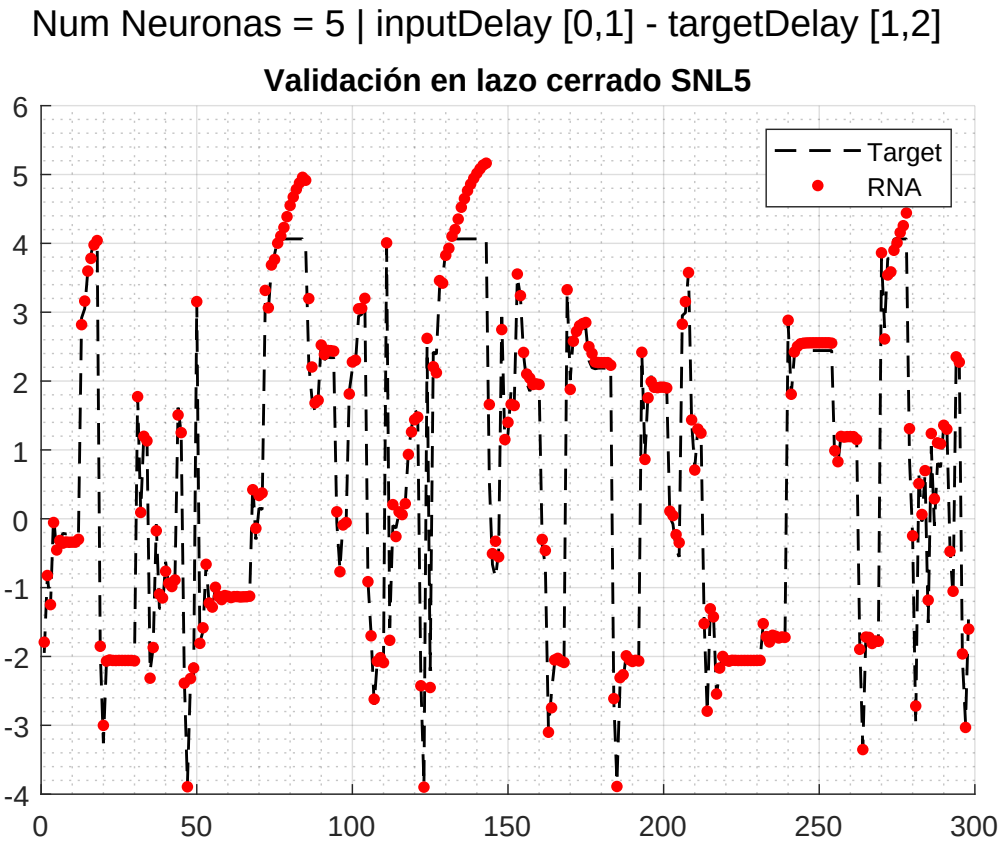

Figura 23: Respuesta del modelo neuronal para el sistema SISO obtenida para validación en lazo cerrado

7.2. Sistema MIMO no-lineal

De la misma forma que para el sistema anterior, el conjunto de datos de entrenamiento se compone por 300 muestras generadas siguiendo la función *camino aleatorio saturado*. Sin embargo, al tratarse de un sistema MIMO con dos entradas y dos salidas, se tienen dos entradas y dos salidas de entrenamiento.

Para ambas entradas, la probabilidad de cambio en cada muestra es del 20 %, igual que antes, pero los cambios son independientes para cada entrada, es decir, no tienen por qué cambiar ambas entradas simultáneamente. Respecto a los rangos de valores empleados, la primera entrada que, al tratarse de un sistema MIMO desacoplado, pertenece al sistema SNL5, toma valores en el mismo rango que en el anterior caso ($[-2{,}5, 2{,}5]$).

Por otro lado, la segunda entrada, correspondiente al sistema SNL1, toma valores dentro del rango $[-1{,}7, 1{,}7]$ para representar las no-linealidades del sistema, pero sin generar entradas muy alejadas de 0, ya que, como se ha dicho, la salida del sistema SNL1 está relacionada con la entrada elevada al cubo.

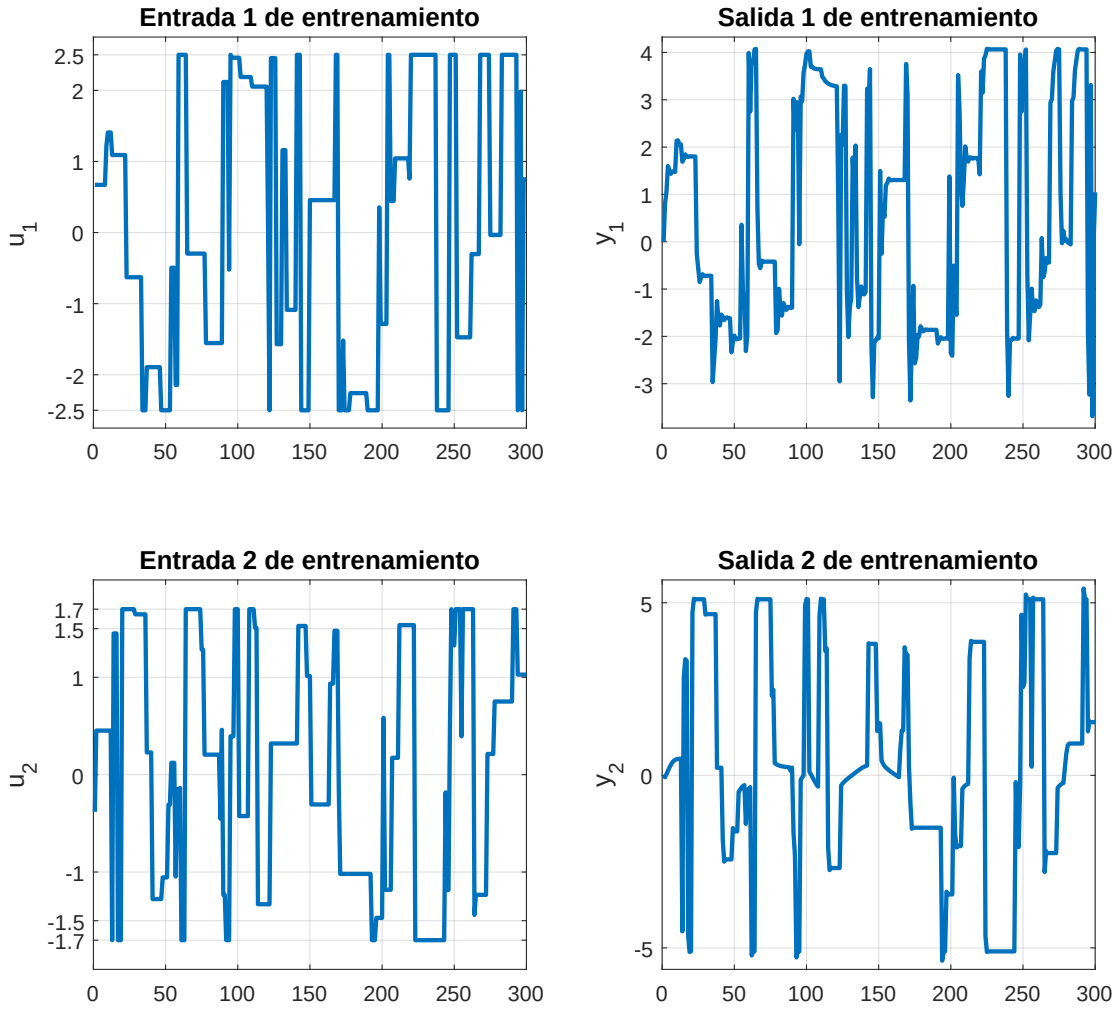

Figura 24: Datos de entrada y salida de entrenamiento para el sistema MIMO no-lineal

A diferencia del conjunto de entrenamiento, se han empleado entradas de validación (Figura 25) con una probabilidad de cambio del 10 %, buscando comprobar en la validación que la red no haya aprendido la tasa de cambio, es decir, que represente el sistema independientemente de que se mantengan los valores de las entradas en el tiempo. El número de muestras y los rangos de valores sí que son iguales a los de entrenamiento, 300 muestras y rangos $[-2{,}5, 2{,}5]$ y $[-1{,}7, 1{,}7]$, respectivamente.

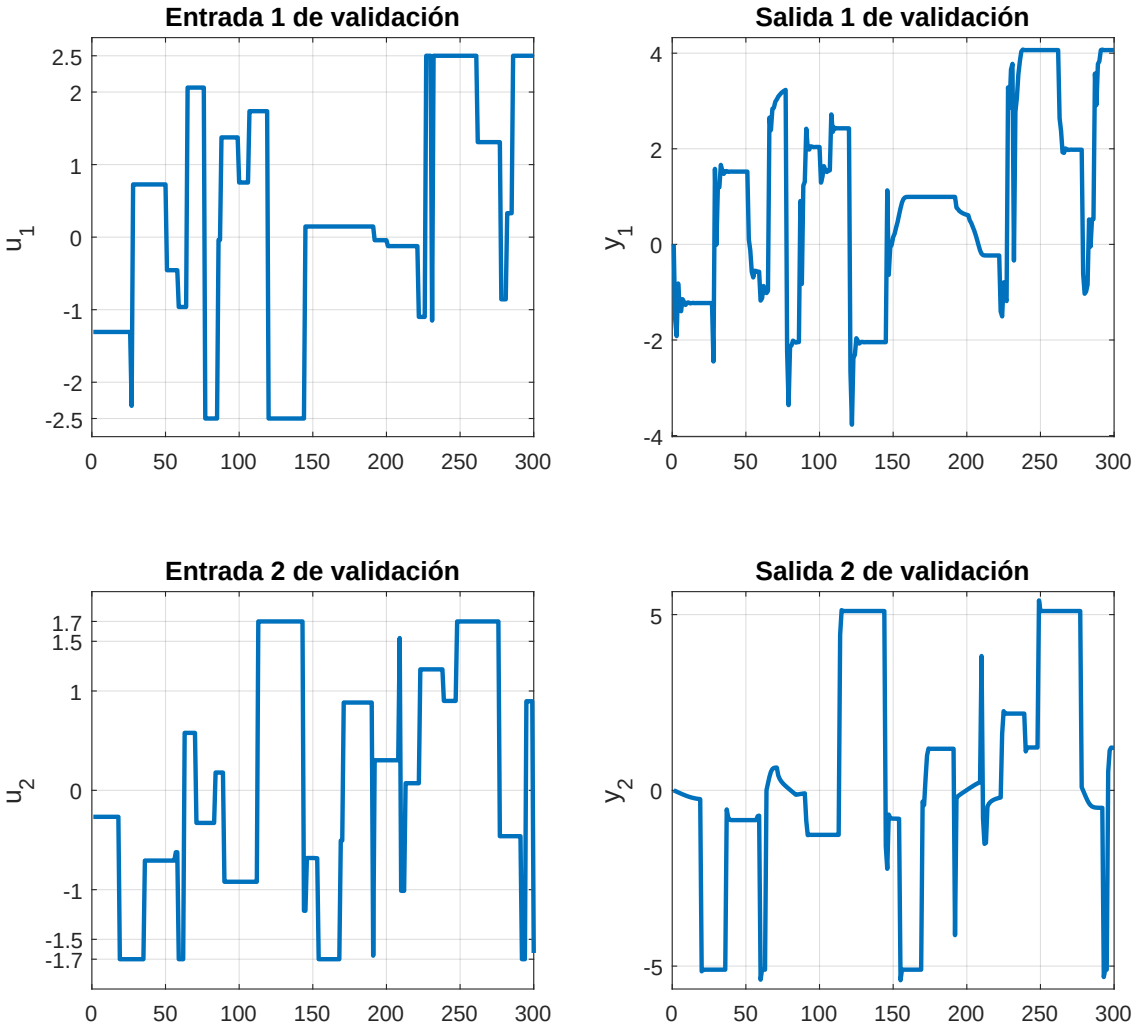

Figura 25: Datos de entrada y salida de validación para el sistema MIMO no-lineal

Una vez generados los conjuntos de entrenamiento y validación, se ha procedido a ejecutar el programa de generación de redes neuronales entrenadas y validadas mediante barrido de parámetros.

Tras generar y procesar la estructura de datos `netData`, se obtienen los gráficos de los rendimientos calculados: error cuadrático medio (MSE) y error absoluto máximo (MaAE) en lazo abierto (Figura 26), y MSE en lazo cerrado (Figura 27).

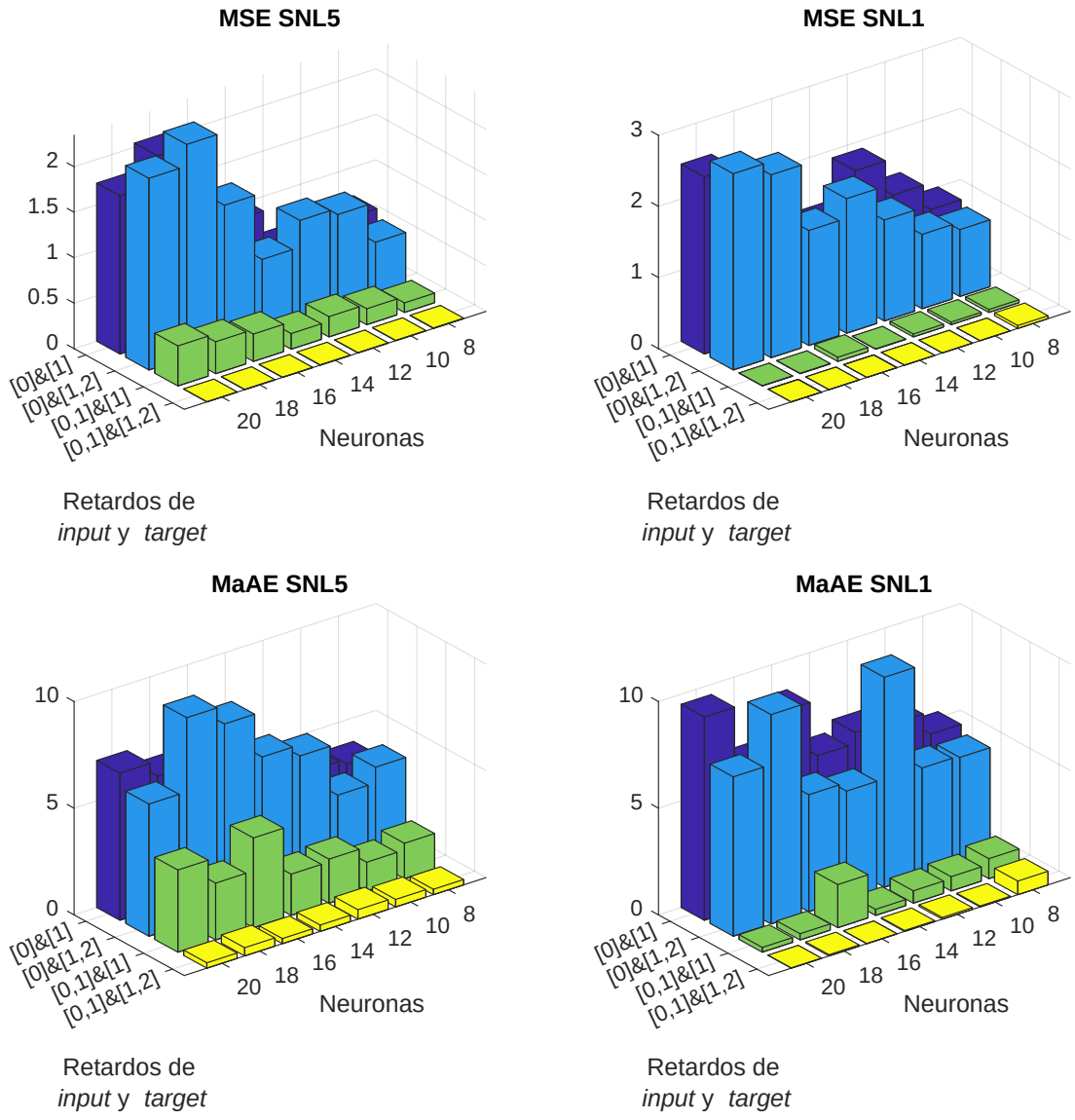

Figura 26: MSE y MaAE en lazo abierto de configuraciones de redes para el sistema MIMO

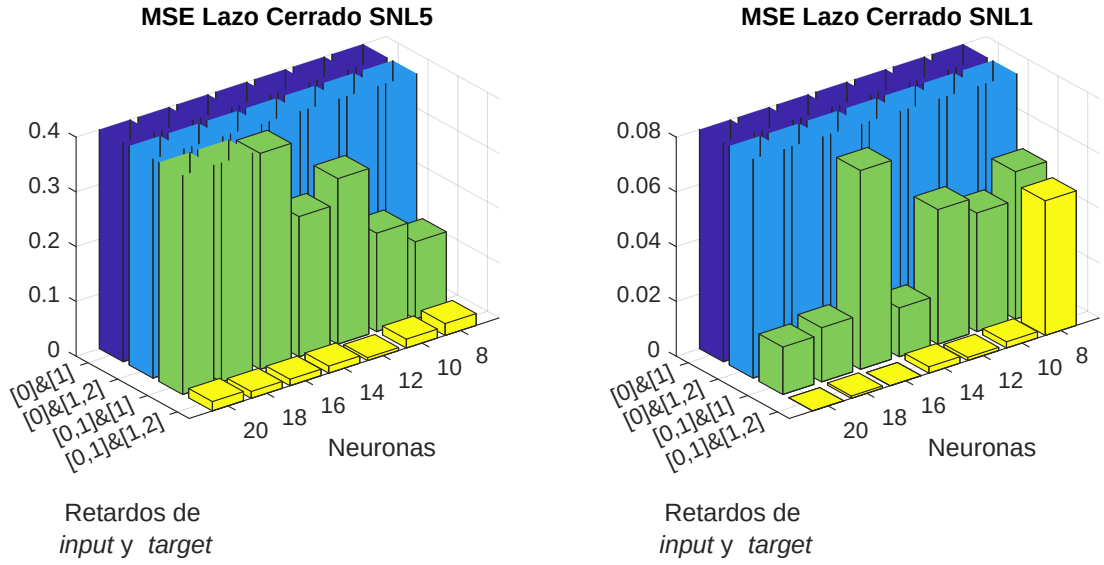

Figura 27: MSE en lazo cerrado de configuraciones de redes para el sistema MIMO

Como se puede comprobar en las Figuras 26 y 27, y de forma similar al anterior sistema objetivo, los parámetros que tienen un mayor impacto en el rendimiento de las redes neuronales son los retardos de *input*.

De hecho, tienen tanto impacto, que en los gráficos relativos a la validación en lazo cerrado (27), se ha tenido que reducir los valores máximos graficados para poder realizar una comparación visual de las mejores redes, y por tanto, las redes neuronales con retardo de *input* 0 (filas azules) quedan fueran de rango.

De todas formas, aunque no se observen bien estos valores, es irrelevante, ya que, en todos ellos, estas redes tienen un peor rendimiento, especialmente en la mencionada validación en lazo cerrado, y en el error cuadrático medio de la salida 2 (correspondiente al sistema SNL1, y por tanto denominada "MSE SNL1"). Por esta razón, estas redes han sido descartadas.

Por otro lado, dentro de las redes con retardos de *input* 0 y 1, aquellas con un único retardo de *target* 1 (fila verde en los gráficos) también han sido descartadas, debido a que, aunque su rendimiento es mucho mejor que las anteriormente mencionadas, en comparación las redes con las redes con retardos de *target* 1 y 2 sus rendimientos son notablemente peores; especialmente en aquellas medidas correspondientes a la primera salida (denominada "SNL5"), donde se observa que los errores cometidos son significativamente mayores.

Esto nos deja con las redes con retardos de *input* 0 y 1, y retardos de *target* 1 y 2 (fila amarilla). De estas, se han preseleccionado para una revisión más a fondo las redes entre 10 y 16 neuronas. La red de 8 neuronas ha sido descartada por su menor rendimiento, mientras que las de 18 y 20 se han descartado por su poca mejora respecto a las redes más pequeñas y, por tanto, con menor coste computacional.

Así, para seleccionar la red más adecuada, se han analizado los datos de rendimiento exactos presentados en las siguientes tablas:

Salida	10 neuronas	12 neuronas	14 neuronas	16 neuronas
Salida 1 (SNL5)	0.0037	0.0035	0.0040	0.0030
Salida 2 (SNL1)	3.231e-04	2.527e-04	3.392e-04	5.595e-07

Tabla 7: MSE en lazo abierto de preselección de redes para el sistema MIMO

Salida	10 neuronas	12 neuronas	14 neuronas	16 neuronas
Salida 1 (SNL5)	0.3508	0.4428	0.2976	0.2386
Salida 2 (SNL1)	0.0429	0.0809	0.0441	0.0038

Tabla 8: MaAE en lazo abierto de preselección de redes para el sistema MIMO

Salida	10 neuronas	12 neuronas	14 neuronas	16 neuronas
Salida 1 (SNL5)	0.0161	0.0054	0.0129	0.0118
Salida 2 (SNL1)	2.267e-03	1.169e-03	2.326e-03	3.786e-06

Tabla 9: MSE en lazo cerrado de preselección de redes para el sistema MIMO

Observando los datos de las Tablas 7, 8, y 9, podemos comprobar que tanto la red con 12 neuronas en la capa oculta, como la red con 16 neuronas, ofrecen rendimientos excelentes. De hecho, ambas opciones son válidas para emplearse como modelos neuronales.

La red con 12 neuronas tiene un menor coste computacional, por lo que podría ser interesante emplearla en algoritmos de control que requieran calcular muchas salidas, como por ejemplo, el ya mencionado control predictivo, que cada instante de tiempo necesita calcular las salidas dentro del horizonte de predicción. Además, su rendimiento en la validación en lazo cerrado ha sido excelente, lo que refuerza esta idea de uso.

Sin embargo, la red elegida ha sido la de 16 neuronas. Esto se debe a que, aunque tiene un mayor coste computacional, si existen los recursos suficientes para poder emplearla, todos sus indicadores de rendimiento en validación son sobresalientes, especialmente aquellos referidos al cálculo de la segunda salida, donde los índices de rendimiento (errores) son órdenes de magnitud menores que los del resto de redes.

Así, el modelo neuronal obtenido para representar este segundo sistema objetivo es una red neuronal con 16 neuronas, retardos de *input* 0 y 1, y retardos de *target* 1 y 2, y por tanto, las entradas de la red son $[u_k, u_{k-1}, y_{k-1}, y_{k-2}]$, teniendo en cuenta que cada uno de estas entradas es un vector de 2 elementos, ya que el sistema tiene 2 entradas (u tiene dos elementos) y 2 salidas (y tiene 2 elementos).

Finalmente, de la misma forma que en el caso del sistema anterior, se han representado las respuestas del modelo neuronal durante la validación en lazo abierto (Figura 28) y en lazo cerrado (Figura 29). Ambas respuestas son notablemente buenas, especialmente la respuesta en lazo abierto, donde apenas de puede observar disparidad entre la salida real y la calculada por el modelo.

En la validación en lazo cerrado, las respuestas también son excelentes, siendo únicamente visible una pequeña diferencia en la primera salida alrededor de la muestra 200, aunque en las muestras anteriores y posteriores el error es mínimo, por lo que se comprueba que las predicciones son buenas y que es capaz de impedir que error se amplifique.

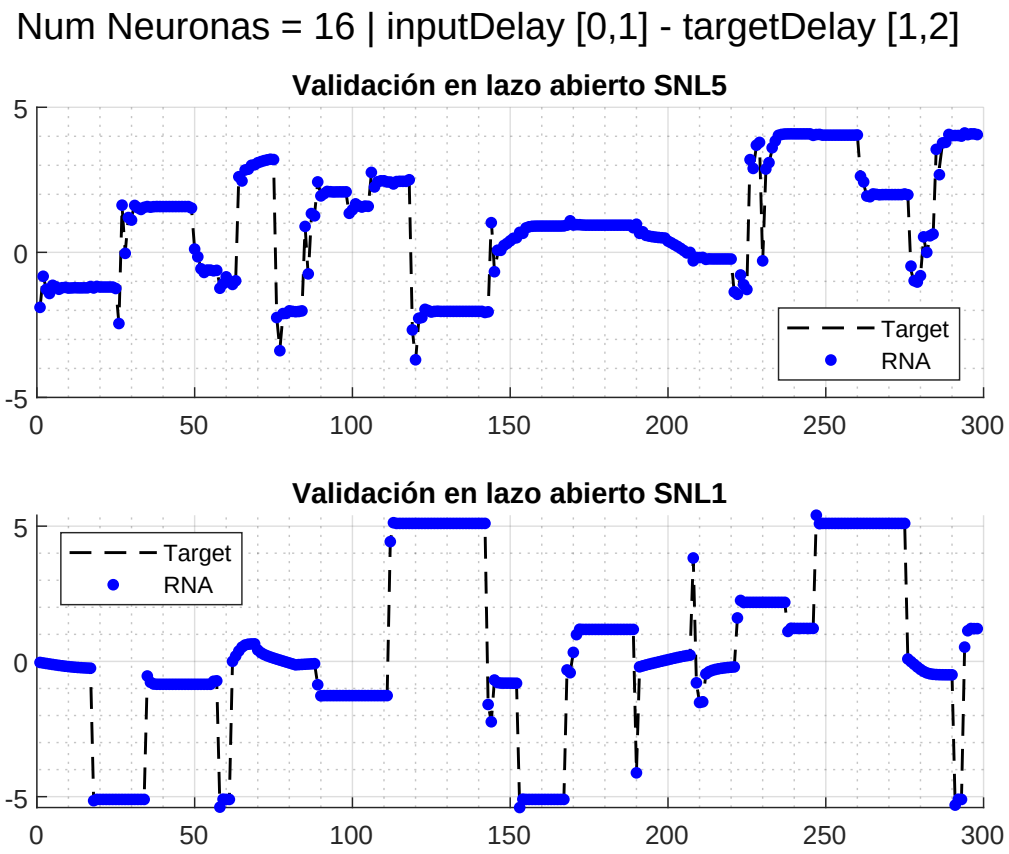

Figura 28: Respuesta del modelo neuronal para el sistema MIMO obtenida para validación en lazo abierto

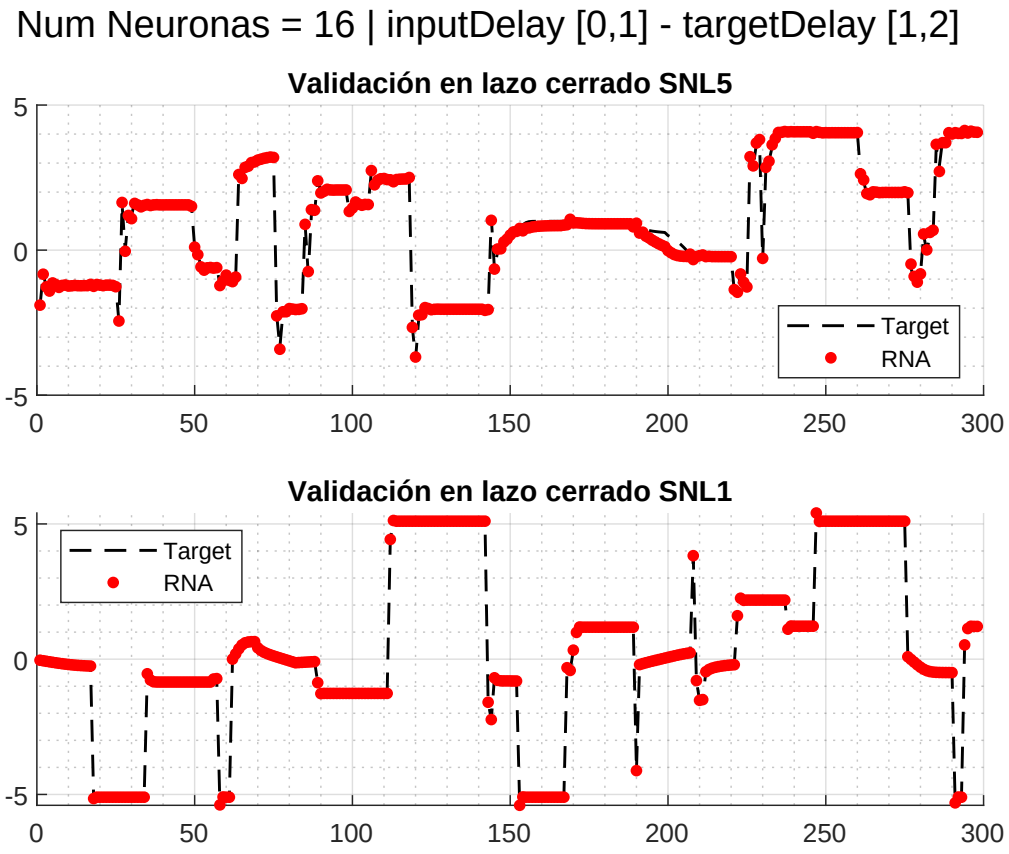

Figura 29: Respuesta del modelo neuronal para el sistema MIMO obtenida para validación en lazo cerrado

8. Plan de proyecto y planificación

En este apartado se presentan el desglose final del trabajo realizado en el proyecto, en fases y tareas. Este es el desarrollo final de la planificación preliminar realizada en el apartado «Objetivos y alcance», y en parte puede considerarse el seguimiento de esta.

Tras ello, se han especificado las fechas de consecución de los hitos especificados, y se ha incluido el diagrama de Gantt correspondiente a la planificación del proyecto. El proyecto ha tenido una duración de 71 días, 7 días más de los 64 previstos en la planificación inicial.

8.1. Descripción de tareas

FASE 1. Fase preliminar

T.1.1: Revisión de la propuesta del TFM

- **Descripción:** Estudio de la propuesta realizada del TFM, con el fin de adquirir una idea clara y concisa de los objetivos y el contexto en el que se enmarca.
- **Duración:** 3 días

FASE 2. Estudio de estado del arte y herramientas

T.2.1: Estado del arte de RNA

- **Descripción:** Búsqueda de información del ámbito de las redes neuronales artificiales y su funcionamiento y estructuras, y de la situación actual en el ámbito.
- **Duración:** 6 días

T.2.2: Estado del arte de identificación de sistemas

- **Descripción:** Búsqueda bibliográfica respectiva a la identificación y modelado de sistemas dinámicos, especialmente el referido a la identificación mediante redes neuronales.
- **Duración:** 5 días

T.2.3: Aprendizaje de herramientas a utilizar

- **Descripción:** Estudio de las herramientas a emplear para el desarrollo del proyecto: MATLAB y su *Deep Learning Toolbox*.
- **Duración:** 3 días

T.2.4: Familiarización con el entorno de trabajo

- **Descripción:** Acercamiento práctico al problema, para lo que se realizan modelos de sistemas simples, con el objetivo de aprender a emplear las funcionalidades de la herramienta.
- **Duración:** 12 días

FASE 3. Desarrollo del trabajo

FASE 3.1. Desarrollo del código

T.3.1.1: Función de generación de entradas

- **Descripción:** Desarrollo del código encargado de generar los conjuntos de datos de entrenamiento y validación mediante *camino aleatorio saturado*.
- **Duración:** 3 días

T.3.1.2: Función de creación y entrenamiento de redes

- **Descripción:** Desarrollo del código encargado de crear y entrenar las redes con parámetros adecuados al trabajo.
- **Duración:** 3 días

T.3.1.3: Función de calculo matricial de salidas

- **Descripción:** Comprender el formato empleado internamente por MATLAB, para desarrollar un código de reducción de redes neuronales y cálculo matricial para disminuir el consumo de memoria y mejorar el rendimiento computacional de los cálculos, respectivamente.
- **Duración:** 5 días

T.3.1.4: Funciones de validación de la red

- **Descripción:** Desarrollo de las funciones encargadas de realizar la validación en lazo abierto y lazo cerrado de los modelos neuronales.
- **Duración:** 4 días

T.3.1.5: Funciones de graficado de resultados

- **Descripción:** Creación de funciones que grafiquen los resultados relevantes del proceso realizado, en un formato adecuado.
- **Duración:** 2 días

FASE 3.2. Sistema SISO no-lineal

T.3.2.1: Estudio del sistema

- **Descripción:** Estudio del sistema SISO no-lineal: ecuación, relaciones de entrada/salida, puntos de operación, no-linealidades....
- **Duración:** 2 días

T.3.2.2: Diseño y entrenamiento de modelos neuronales

- **Descripción:** Generación de los modelos neuronales para el sistema SISO no-lineal mediante el código desarrollado.
- **Duración:** 2 días

T.3.2.3: Validación de modelos neuronales

- **Descripción:** Validación en lazo abierto y cerrado de los modelos neuronales para el sistema SISO no-lineal, comparación, y selección del modelo neuronal final.
- **Duración:** 3 días

FASE 3.3. Sistema MIMO no-lineal

T.3.3.1: Estudio del sistema

- **Descripción:** Estudio del sistema MIMO no-lineal: ecuaciones, relaciones de entrada/salida, puntos de operación, no-linealidades....
- **Duración:** 3 días

T.3.3.2: Diseño y entrenamiento de modelos neuronales

- **Descripción:** Generación de los modelos neuronales para el sistema MIMO no-lineal mediante el código desarrollado.
- **Duración:** 3 días

T.3.3.3: Validación de modelos neuronales

- **Descripción:** Validación en lazo abierto y cerrado de los modelos neuronales para el sistema MIMO no-lineal, comparación, y selección del modelo neuronal final.
- **Duración:** 4 días

FASE 3.4. Integración y simplificación del código

T.3.4.1: Integración del código

- **Descripción:** Integración de las funciones desarrolladas en funciones mayores que las implementen de forma que se optimice el proceso, para lo que se ha empleado el barrido de parámetros.
- **Duración:** 5 días

T.3.4.2: Simplificación y modularización del código

- **Descripción:** Últimas modificaciones al código para facilitar su uso y comprensión, con el objetivo de ayudar a optimizar el desarrollo de futuros modelos neuronales.
- **Duración:** 3 días

FASE 4. Documentación

T.4.1: Documentación del proyecto

- **Descripción:** Redacción del documento recopilando toda la información del trabajo realizado en el TFM.
- **Duración:** 71 días

8.2. Hitos

Hito	Descripción	Fecha
H1	Código de creación y validación de modelos neuronales completado	Semana 10
H2	Modelo neuronal del sistema SISO no-lineal generado	Semana 13
H3	Modelo neuronal del sistema MIMO no-lineal generado	Semana 15
H4	Código de desarrollo de modelos neuronales integrado: proyecto finalizado	Semana 17

Tabla 10: Hitos finales del proyecto

8.3. Diagrama de Gantt

Tarea	Nombre de tarea	Duración	Comienzo	Fin
F.1	**Fase preliminar**	**3 días**	**23/05/2022**	**25/05/2022**
T.1.1	Revisión de la propuesta del TFM	3 días	23/05/2022	25/05/2022
F.2	**Estudio de estado del arte y herramientas**	**26 días**	**26/05/2022**	**30/06/2022**
T.2.1	Estado del arte de RNA	6 días	26/05/2022	02/06/2022
T.2.2	Estado del arte de identificación de sistemas	5 días	03/06/2022	09/06/2022
T.2.3	Aprendizaje de herramientas a utilizar	3 días	10/06/2022	14/06/2022
T.2.4	Familiarización con el entorno de trabajo	12 días	15/06/2022	30/06/2022
F.3	**Desarrollo del trabajo**	**42 días**	**01/07/2022**	**12/09/2022**
F.3.1	**Desarrollo del código**	**17 días**	**01/07/2022**	**25/07/2022**
T.3.1.1	Función de generación de entradas	3 días	01/07/2022	05/07/2022
T.3.1.2	Función de creación y entrenamiento de redes	3 días	06/07/2022	08/07/2022
T.3.1.3	Función de cálculo matricial de salidas	5 días	11/07/2022	15/07/2022
T.3.1.4	Funciones de validación de la red	4 días	18/07/2022	21/07/2022
T.3.1.5	Funciones de graficado de resultados	2 días	22/07/2022	25/07/2022
H1	Código desarrollado	0 días	25/07/2022	25/07/2022
F.3.2	**Sistema SISO no-lineal**	**7 días**	**26/07/2022**	**17/08/2022**
T.3.2.1	Estudio del sistema	2 días	26/07/2022	27/07/2022
T.3.2.2	Diseño y entrenamiento de modelos neuronales	2 días	28/07/2022	29/07/2022
T.3.2.3	Validación de modelos neuronales	3 días	15/08/2022	17/08/2022
H2	Sistema SISO no-lineal modelado	0 días	17/08/2022	17/08/2022
F.3.3	**Sistema MIMO no-lineal**	**10 días**	**18/08/2022**	**31/08/2022**
T.3.3.1	Estudio del sistema	3 días	18/08/2022	22/08/2022
T.3.3.2	Diseño y entrenamiento de modelos neuronales	3 días	23/08/2022	25/08/2022
T.3.3.3	Validación de modelos neuronales	4 días	26/08/2022	31/08/2022
H3	Sistema MIMO no-lineal modelado	0 días	31/08/2022	31/08/2022
F.3.4	**Integración y simplificación del código**	**8 días**	**01/09/2022**	**12/09/2022**
T.3.4.1	Integración del código	5 días	01/09/2022	07/09/2022
T.3.4.2	Simplificación y modularización del código	3 días	08/09/2022	12/09/2022
H4	Código integrado	0 días	12/09/2022	12/09/2022
F.4	**Documentación**	**71 días**	**23/05/2022**	**12/09/2022**
T.4.1	Documentación del proyecto	71 días	23/05/2022	12/09/2022

Tabla 11: Planificación final del proyecto

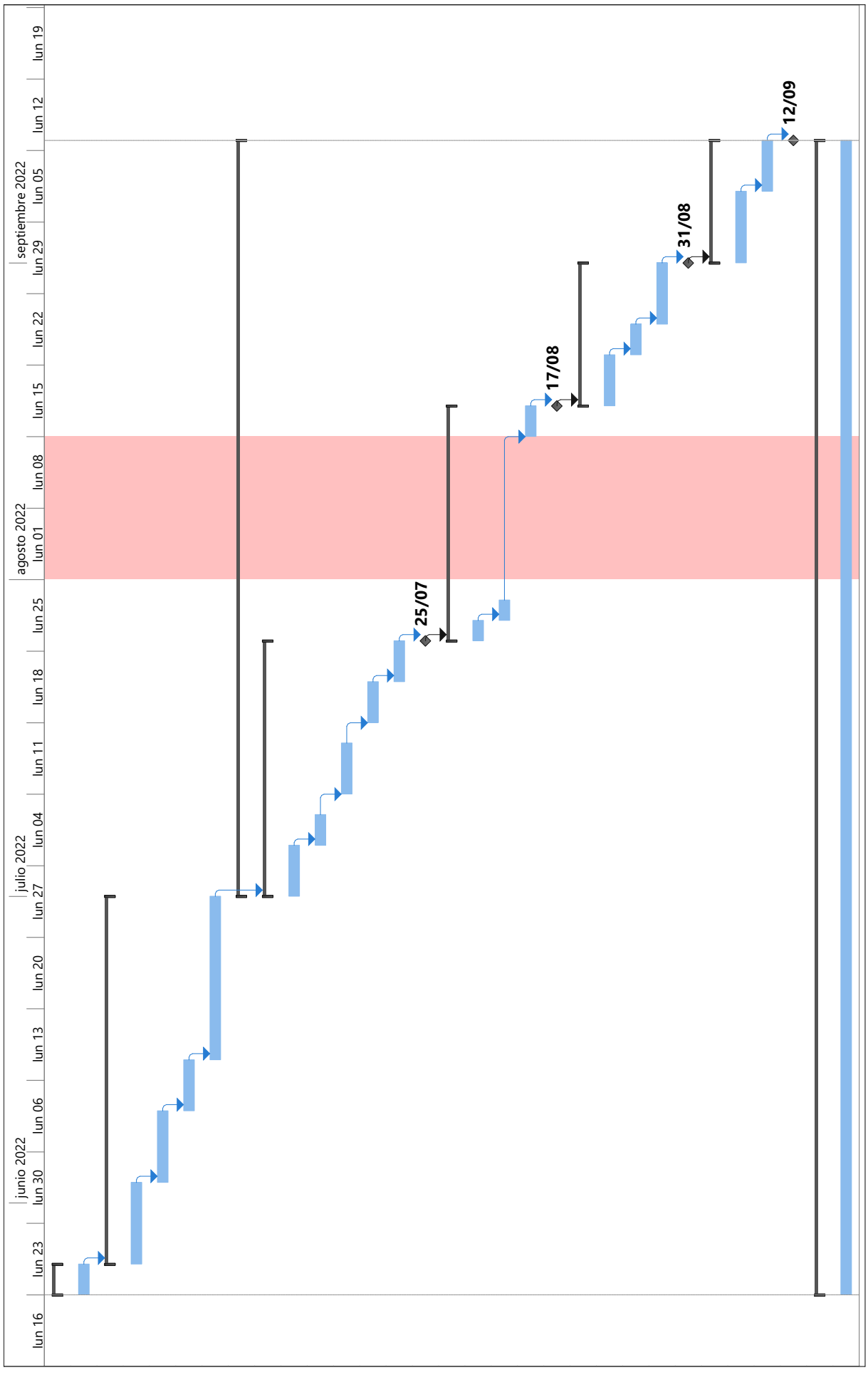

Figura 30: Diagrama de Gantt del proyecto

9. Descripción del presupuesto

En este apartado se aborda la realización del presupuesto final usado para realizar el proyecto. Respecto a las diversas consideraciones antes de realizar el presupuesto, se ha seguido el mismo criterio que en el presupuesto provisional del apartado «Objetivos y alcance»:

- Se ha estimado la tasa horaria del alumno como la de un ingeniero júnior (20 €/h).
- El alumno trabaja todos los días durante 8 horas diarias, durante toda la duración del proyecto, en este caso, igual a 71 días.
- El periodo de amortización del *hardware* (PC) es de 5 años.
- El periodo de amortización del *software* (PC) es de 3 años.

Por lo demás, no se ha incurrido en gastos (fungibles), y se estima que los costes indirectos adicionales son del 4 % de los costes directos. Adicionalmente, se debe añadir un 2 % correspondiente al coste de capital.

De esta forma, el presupuesto total del proyecto resultante es de 12.681,75 €. En la Tabla 13 se puede observar el presupuesto con más detalle.

Comparando con el presupuesto inicial realizado, el presupuesto final es muy cercano al estimado, aunque cabe destacar que la partida de imprevistos provista es imprescindible para que esto haya sido así. Esto es debido a que todos los costes, y especialmente el de horas internas, que es la gran mayoría de estos, es proporcional al número de horas trabajadas.

Al haberse extendido el proyecto de los 64 días previstos a 71, estos gastos han aumentado en la misma proporción, que es cercana al 10 % adicional. De esta forma, al coincidir este porcentaje con el porcentaje reservado a imprevistos, los presupuestos resultan muy similares.

Cálculo de tasas horarias			
	Coste (€)	Vida útil (h)	Tasa horaria (€/h)
PC	2.000	9.000	0,222
MATLAB R2022a	2.100	5.400	0,389
Deep Learning Toolbox	1.200	5.400	0,222
Parallel Computing Toolbox	1.050	5.400	0,194
Control System Toolbox	1.200	5.400	0,222

Tabla 12: Cálculo de las tasas horarias del proyecto

HORAS INTERNAS	Cantidad (h)	Tasa horaria (€/h)	Coste (€)	**11.360,00 €**
Alumno	568	20,000	11.360,00	
AMORTIZACIONES	Cantidad (h)	Tasa horaria (€/h)	Coste (€)	**594,89 €**
PC	568	0,222	126,22	
MATLAB R2022a	456	0,389	177,33	
Deep Learning Toolbox	456	0,222	101,33	
Parallel Computing Toolbox	456	0,194	88,67	
Control System Toolbox	456	0,222	101,33	
COSTES DIRECTOS				**11.954,89 €**
Costes indirectos	4 %		478,20 €	
SUBTOTAL				**12.433,08 €**
Coste de capital	2 %		248,66 €	
TOTAL				**12.681,75 €**

Tabla 13: Presupuesto final del proyecto

10. Conclusiones y trabajos futuros

En este trabajo se ha desarrollado los modelos neuronales de dos sistemas no-lineales complejos, principalmente orientados a su uso como modelos en estrategias de control predictivo, especialmente en la estrategia iMO-NMPC; y se ha estudiado su rendimiento. Por otro lado, se ha desarrollado un código de entrenamiento y validación de modelos neuronales para optimizar el desarrollo de modelos futuros.

Para esto, se ha investigado y detallado el ámbito general en que se enmarca el trabajo y los avances en este, el de las redes neuronales artificiales. De esta forma, se ha comprobado que las redes neuronales son una gran opción a la hora de desarrollar modelos de caja negra de sistemas de diversos campos, y se han comparado distintas estructuras de RNA, teniendo en cuenta las ventajas e inconvenientes de cada una de ellas.

Una vez contextualizado el trabajo, se han explicado y justificado las diversas consideraciones y elecciones hechas durante el desarrollo del proyecto para enmarcar el entorno concreto en el que se ha trabajado, especificando la estructura de red neuronal y el algoritmo de entrenamiento empleados, entre otros.

De esta forma, una vez reducido el amplio espectro de posibilidades dentro de este campo, se ha descrito y detallado el trabajo realizado, y el proceso seguido durante el desarrollo del trabajo.

En este proceso, se ha prestado especial atención a la disminución del coste computacional de las redes neuronales, por un doble motivo: el menor uso de recursos durante su entrenamiento y aplicación, y debido al riesgo identificado durante el análisis de riesgos.

Por esta razón, se ha desarrollado un código de cálculo matricial de salidas de la red neuronal, mediante el cual se ha reducido el coste computacional del desarrollo de los modelos neuronales en este trabajo, reduciendo así el tiempo empleado para ello.

Adicionalmente, este desarrollo, que no estaba previsto en la planificación inicial, no ha causado retrasos adicionales, ya que se estima que el tiempo empleado en desarrollar este código se ha ahorrado durante el entrenamiento y validación de los modelos neuronales.

Sin embargo, aunque esto no ha supuesto retrasos adicionales, el proyecto sí que ha tenido un retraso de 7 días debido a imprevistos y la mayor duración de algunas tareas, lo que ha causado que los costes directos del proyecto hayan sido mayores de los calculados. Aún así, este posible retraso y mayor coste se había contemplado en el presupuesto inicial, quedando cubierto con la partida reservada a imprevistos.

En el ámbito de los resultados obtenidos, se ha comprobado el correcto funcionamiento de estos modelos neuronales en pruebas de validación, tanto en lazo abierto, como en lazo cerrado, obteniendo buenos resultados.

En este proceso también se ha tenido muy en cuenta la reducción del coste computacional, ofreciendo para cada sistema un modelo alternativo con menos neuronas, pero con buen rendimiento.

En este respecto, los trabajos futuros en la línea de investigación pasan por la identificación de sistemas aún más complejos, una vez establecidas las bases para un desarrollo más eficiente mediante el código desarrollado. Esto pasa por el modelado de sistemas multivariables más complejos, con la adición de acoplamientos, o dinámicas aún más complejas. También puede pasar por el desarrollo de modelos de sistemas reales.

Por otro lado, tras el desarrollo de los modelos neuronales realizado en este trabajo, queda realizar la implementación de estos en la estrategia de control iMO-NMPC, primero en simulación, pero especialmente en sistemas empotrados de tiempo real, como Speedgoat, donde la memoria y el poder de cómputo son factores determinantes.

En este aspecto, el énfasis puesto en la reducción del coste computacional resulta particularmente útil para mejorar el rendimiento, mediante el cálculo matricial y la búsqueda de redes de tamaño reducido.

Finalmente, el estudio de la implementación del entrenamiento *on-line* puede resultar interesante, con el objetivo de añadir una capa de adaptabilidad al control, con el objetivo de diseñar un sistema de control efectivo y potente.

Bibliografía

[1] K. Narendra y K. Parthasarathy, «Identification and control of dynamical systems using neural networks,» IEEE Transactions on Neural Networks, vol. 1, n.º 1, págs. 4-27, 1990. DOI: 10.1109/72.80202.

[2] M. M. Polycarpou y P. A. Ioannou, «Identification and Control of Nonlinear Systems Using Neural Network Models: Design and Stability Analysis,» ELECTRICAL ENGINEERING—SYSTEMS REP, inf. téc., 1991. dirección: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.1816.

[3] J. J. Valera García, V. Gómez Garay, E. Irigoyen Gordo, F. Artaza Fano y M. Larrea Sukia, «Intelligent Multi-Objective Nonlinear Model Predictive Control (iMO-NMPC): Towards the 'on-line' optimization of highly complex control problems,» Expert Systems with Applications, vol. 39, n.º 7, págs. 6527-6540, 2012, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2011.12.052.

[4] K. Hornik, M. Stinchcombe y H. White, «Multilayer feedforward networks are universal approximators,» Neural Networks, vol. 2, n.º 5, págs. 359-366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8.

[5] G. Cybenko, «Approximation by superpositions of a sigmoidal function,» Mathematics of Control, Signals and Systems, vol. 2, n.º 4, págs. 303-314, dic. de 1989, ISSN: 1435-568X. DOI: 10.1007/BF02551274.

[6] J. Deng, S. Sierla, J. Sun y V. Vyatkin, «Reinforcement learning for industrial process control: A case study in flatness control in steel industry,» Computers in Industry, vol. 143, pág. 103 748, 2022, ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2022.103748.

[7] M. Hagan y M. Menhaj, «Training feedforward networks with the Marquardt algorithm,» IEEE Transactions on Neural Networks, vol. 5, n.º 6, págs. 989-993, nov. de 1994, ISSN: 1941-0093. DOI: 10.1109/72.329697.

[8] J. Sjöberg, H. Hjalmarsson y L. Ljung, «Neural Networks in System Identification,» IFAC Proceedings Volumes, vol. 27, n.º 8, págs. 359-382, 1994, IFAC Symposium on System Identification (SYSID'94), Copenhagen, Denmark, 4-6 July, ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)47737-8.

[9] H. Song, X. Shan, L. Zhang, G. Wang y J. Fan, «Research on identification and active vibration control of cantilever structure based on NARX neural network,» Mechanical Systems and Signal Processing, vol. 171, pág. 108 872, 2022, ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2022.108872.

[10] Q. Liu, W. Chen, H. Hu, Q. Zhu y Z. Xie, «An Optimal NARX Neural Network Identification Model for a Magnetorheological Damper With Force-Distortion Behavior,» Frontiers in Materials, vol. 7, 2020, ISSN: 2296-8016. DOI: 10.3389/fmats.2020.00010.

[11] Z. Boussaada, O. Curea, R. Ahmed, H. Camblong Ruiz y N. Mrabet Bellaaj, «A Nonlinear Autoregressive Exogenous (NARX) Neural Network Model for the Prediction of the Daily Direct Solar Radiation,» Energies, vol. 11, pág. 620, mar. de 2018. DOI: 10.3390/en11030620.

[12] A. Bamimore, A. Osinuga, T. Kehinde-Abajo, A. Osunleke y O. Taiwo, «A Comparison of Two Artificial Neural Networks for Modelling and Predictive Control of a Cascaded Three-Tank System,» IFAC-PapersOnLine, vol. 54, n.º 21, págs. 145-150, 2021, Control Conference Africa CCA 2021, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2021.12.025.

[13] M. Larrea, E. Larzabal, E. Irigoyen, J. Valera y M. Dendaluce, «Implementation and testing of a soft computing based model predictive control on an industrial controller,» Journal of Applied Logic, vol. 13, n.º 2, Part A, págs. 114-125, 2015, SI: SOCO12, ISSN: 1570-8683. DOI: https://doi.org/10.1016/j.jal.2014.11.005.

[14] C. J. Harris, Advances in Intelligent Control. USA: Taylor & Francis, Inc., 1994, ISBN: 0748400664.

Anexo I: Código y programas fuente

Funciones principales

Código 1: Función de barrido de parámetros: generación de estructura de datos

```
function netData_out = NARX_paramSweep(numNeur,inDel,tgDel,numRep,...
                                                        trData,valData)

%% Inicialización de la celda de datos
% Crear de combinaciones de retardos (fila 1: inputs, fila 2: targets)
delays = cell(2,length(inDel)*length(tgDel));
delays(1,:) = repelem(inDel,length(tgDel));
delays(2,:) = repmat(tgDel,1,length(inDel));

% Inicialización de celda de datos
numDataElements = length(numNeur)*length(inDel)*length(tgDel)*numRep;
netData = cell(length(numNeur)+2,size(delays,2)+1);

% Etiquetado de filas y columnas (mediante subfunción más abajo)
netData = tag_netData(netData,numNeur,delays);

%% Bucle principal
% Bucles for anidados para barrer todas las posibilidades
for i=1:length(numNeur)
    for j=1:size(delays,2)
        % Inicializar array de estrcturas
        netStruct = struct('net',cell(1,numRep),...
                           'trainPerf',cell(1,numRep),...
                           'openVal_out',cell(1,numRep),...
                           'openVal_mse',cell(1,numRep),...
                           'openVal_maxabse',cell(1,numRep),...
                           'closedVal_out',cell(1,numRep),...
                           'closedVal_mse',cell(1,numRep));
        for k=1:numRep
            % Número de iteración, para mostrar progreso en pantalla
            numIter = (size(delays,2)*numRep)*(i-1)+numRep*(j-1)+k;

            % Entrenar la red
            [net, trainRec] = NARX_trainNet(trData.input,...
                    trData.target,numNeur(i),delays{1,j},delays{2,j});

            % Validar la red en lazo abierto
            [openVal_out, openVal_mse, openVal_maxabse] = ...
                    NARX_val_open(net,valData.input,valData.target);

            % Validar la red en lazo cerrado
            [closedVal_out, closedVal_mse] = ...
                    NARX_val_closed(net,valData.input,valData.target);

            % Mostrar en pantalla progreso del programa
            fprintf("Entrenamiento y validación completada para " + ...
                    "elemento {%u,%u,%u} (%u/%u)\n",...
                    i,j,k,numIter,numDataElements);

            % Estructura guardada en cada elemento de la celda, contiene
            % diversa información:
            %  - Red neuronal
            %  - Rendimiento de entrenamiento
```

```
            %  - Datos de validación en lazo abierto (salida, MSE, MaAE)
            %  - Datos de validación en lazo cerrado (salida, MSE)
            netStruct(k) = struct('net',net,...
                                  'trainPerf',trainRec.best_perf,...
                                  'openVal_out',openVal_out,...
                                  'openVal_mse',openVal_mse,...
                                  'openVal_maxabse',openVal_maxabse,...
                                  'closedVal_out',closedVal_out,...
                                  'closedVal_mse',closedVal_mse);
        end % bucle for k
        % Guardar array de estructuras en celda
        netData{i,j} = netStruct;
    end % bucle for j
end % bucle for i

% Asignar salida y mostrar texto de finalización del programa
netData_out = netData;
disp("Barrido finalizado: redes entrenadas y validadas");
end % function

%% Subfunción de etiquetado de celda de datos
function netData_tagged = tag_netData(netData,numNeur,delays)
    % Crear etiquetas de columnas
    for i=1:(size(netData,2)-1)
        netData{end-1,i} = strcat("In: ",num2str(delays{1,i}));
        netData{end,i} = strcat("Tg: ",num2str(delays{2,i}));
    end

    % Crear celda de etiquetas de filas
    for i=1:(size(netData,1)-2)
        netData{i,end} = strcat("Neuronas: ",num2str(numNeur(i)));
    end

    % Asignar salida
    netData_tagged = netData;
end % subfunction
```

Código 2: Función de procesado de estructura de datos

```
function netData_proc_out = NARX_process_netData(varargin)

%% Comprobar parámetros de entrada y asignar a variables
% Comprobar número de parámetros de entrada
if (nargin < 1) || (nargin > 2)
    error("Número inválido de entradas: debe ser 1 or 2");
end

% Asignar entradas a variables
% varargin{1} -> netData
% varargin{2} -> mode (formato string, si es formato array de
%                      caracteres,cambiar a string) (por defecto, "mse")

% Asignar netData
netData = varargin{1};

% Asignar mode
if nargin >= 2
    if isstring(varargin{2})
        mode = varargin{2};
    elseif ischar(varargin{2})
        mode = convertCharsToStrings(varargin{2});
    else
        error("La segunda entrada debe ser un array de ..." + ...
            "caracteres o una string");
    end % if
    if ~((mode == "mse") || (mode == "maxabse"))
        error("La segunda entrada debe ser ""mse"" or ""maxabse""");
    end
else
    % Caso por defecto: "mse"
    mode = "mse";
end % if

%% Inicializar celda y etiquetas
netData_proc = cell(size(netData));
netData_proc(end-1:end,1:end-1) = netData(end-1:end,1:end-1,1);
netData_proc(1:end-2,end) = netData(1:end-2,end,1);

%% Extraer mejor red neuronal para cada caso
% Tamaño de los datos (menor debido a los elementos que contienen las
% etiquetas)
sizeData = size(netData)-[2,1];

% Bucle principal
for i=1:sizeData(1)
    for j=1:sizeData(2)
        % Buscar índice correspondiente al menor error (según se haya
        % especificado y ponderado en los parámetros de entrada)
        if mode == "mse"
            mseCheck = sum([netData{i,j}.openVal_mse].^2,1);
            [~,index] = min(mseCheck,[],'all');
        elseif mode == "maxabse"
            maxabseCheck = sum([openVal_data.maxabse].^2,1);
            [~,index] = min(maxabseCheck,[],'all');
        end % if
```

```
57
58          % Obtener rendimientos medios del conjunto de redes neuronales
59          mean_openVal_mse = mean([netData{i,j}.openVal_mse]);
60          mean_openVal_maxabse = mean([netData{i,j}.openVal_maxabse]);
61          mean_closedVal_mse = mean([netData{i,j}.closedVal_mse]);
62
63          % Extraer información de la mejor red neuronal:
64          %  - Mejor red neuronal y sus datos
65          %  - Media de los MSE para validación en lazo abierto
66          %  - Media de los MaAE para validación en lazo abiertoç
67          %  - Media de los MaAE para validación en lazo cerrado
68          netData_proc{i,j} = struct('bestNet',netData{i,j}(index),...
69                          'mean_openVal_mse',mean_openVal_mse,...
70                          'mean_openVal_maxabse',mean_openVal_maxabse,...
71                          'mean_closedVal_mse',mean_closedVal_mse);
72      end % bucle for j
73  end % bucle for i
74
75  % Asignar salida y mostrar texto de finalización del programa
76  netData_proc_out = netData_proc;
77  disp("Estructura de datos procesada");
78  end % function
```

Código 3: Función de graficado y guardado de imágenes

```
function NARX_plot_netData(netData,valData,figSavePath)

%% Si no existe la carpeta para guardar las imágenes, la crea
if ~exist(figSavePath,'dir')
    mkdir(figSavePath);
end

%% Inicializar datos y preparar celdas con nombres
% Para nombrar redes, etiquetas de datos en ejes, títulos de gráficos...

% Tamaño de datos de salida (ignorar etiquetas) y número de salidas
% (igual al número de error).
sizeData = size(netData)-[2,1];
numOutputs = length(netData{1,1}.bestNet.openVal_mse);

% Crear celdas con número de neuronas para nombrar cada red y para usar
% como etiquetas de datos en ejes
numNeurons = cell(1,sizeData(1));
for i=1:sizeData(1)
    numNeurons{i} = num2str(size(netData{i,1}.bestNet.net.IW{1,1},1));
end

% Crear celdas con retardos de entradas para nombrar cada red y para
% usar como etiquetas de datos en ejes
inDelays = cell(1,sizeData(2));
tgDelays = cell(1,sizeData(2));
numDelays = cell(1,sizeData(2));
for i=1:sizeData(2)
    inputDelays = netData{1,i}.bestNet.net.inputWeights{1,1}.delays;
    targetDelays = netData{1,i}.bestNet.net.inputWeights{1,2}.delays;
    inDelays{i} = strrep(num2str(inputDelays),'  ',',');
    tgDelays{i} = strrep(num2str(targetDelays),'  ',',');
    numDelays{i} = strcat("[",strrep(num2str(inputDelays),'  ',','), ...
                    "]&[",strrep(num2str(targetDelays),'  ',','),"]");
end

% Nombres de las salidas, para nombrar los gráficos
if numOutputs == 1
    outputNames = "SNL5";
else
    outputNames = ["SNL5","SNL1"];
end

%% Bucle de gráficos de validación
% Inicializar figura de gráficos de validación
h_fig = figure(42);
set(h_fig,'Color',[1 1 1],'Position',[216 418 560 420], ...
                                        'renderer','painters');
pause(0.05);

% Bucle principal de gráficos
for i=1:sizeData(1)
    for j=1:sizeData(2)
        for valType=["open","closed"] % Hacer un gráfico para cada val
            % Obtener retardo máximo de la red y datos de validación
            maxDelay = netData{i,j}.bestNet.net.numInputDelays;
```

```
            if valType == "open"
                val_out = netData{i,j}.bestNet.openVal_out;
                lineType = 'b.';
            else
                val_out = netData{i,j}.bestNet.closedVal_out;
                lineType = 'r.';
            end

            % Etiquetas de número de neuronas y retardos (para los
            % títulos de los gráficos), y nombre del archivo a generar
            tagNeurons = strcat("N = ",numNeurons{i});
            tagDelays = strcat("inDel [",inDelays{j},"] - tgDel [", ...
                                                    tgDelays{j},"]");
            netName = strcat("net-Num Neur",numNeurons{i},"-iD", ...
                                strrep(inDelays{j},',',''),"-tD", ...
                                strrep(tgDelays{j},',',''),'.fig');

            % Poner título al gráfico
            sgtitle(strcat(tagNeurons," | ",tagDelays));

            % Graficar cada salida en un subplot
            for graphOutput=1:numOutputs
                subplot(numOutputs,1,graphOutput);
                hold on;
                plot(valData.target(graphOutput,maxDelay+1:end),...
                                                'k--','LineWidth',1);
                plot(val_out(graphOutput,:),lineType,'MarkerSize',10);
                hold off;
                grid;
                grid minor;
                title(strcat("Validación ",outputNames{graphOutput}));
                legend("Target","RNA");
            end % bucle for graphOutput

            % Guardar figura en archivo y limpiar figura (para evitar
            % crear múltiples figuras)
            savefig(h_fig,figSavePath+netName+" - "+valType);
            pause(0.05);
            clf;
        end % bucle for valType
    end % bucle for j
end % bucle for i

% Cerrar figura
close(42);

%% Graficar errores en gráficos de barras
% Crear matrices de errores
netData_mat = cell2mat(netData(1:sizeData(1),1:sizeData(2)));
bestNet_mat = [netData_mat.bestNet];

open_mse_mat = reshape([bestNet_mat.openVal_mse]', ...
                                    sizeData(1),sizeData(2),[]);
open_maxabse_mat = reshape([bestNet_mat.openVal_maxabse]', ...
                                    sizeData(1),sizeData(2),[]);
closed_mse_mat = reshape([bestNet_mat.closedVal_mse]', ...
                                    sizeData(1),sizeData(2),[]);

```

```
% Inicializar objeto gráfico y número de plot (para generar figuras
% distintas)
h_mse_maxabse = gobjects(1,3*numOutputs);
plotNum = 0;

% Bucle de gráficos de barras
for errorType = ["MSE","MaAE","MSE Lazo Cerrado"]
    % Elegir matriz de error MSE, MaAE, o divergencia, según el gráfico
    switch errorType
        case "MSE"
            plotMat = open_mse_mat;
        case "MaAE"
            plotMat = open_maxabse_mat;
        case "MSE Lazo Cerrado"
            plotMat = closed_mse_mat;
    end % switch

    % Para los errores de todas las salidas, realizar gráfico de barras
    % y etiquetarlo adecuadamente
    for i=1:size(plotMat,3) % = numOut
        plotNum = plotNum + 1;
        h_mse_maxabse(plotNum) = figure(42+plotNum);

        bar3(plotMat(:,:,i))
        view(52.5,25);
        title(strcat(errorType," ",outputNames{i}));
        xlabel("Retardos de {\it input} y {\it target}");
        xticklabels(numDelays);
        ylabel("Neuronas");
        yticklabels(numNeurons);
    end % bucle for i
end % bucle for errorType

% Guardar todas las figuras en un único archivo.fig, y cerrar figuras
% generadas
savefig(h_mse_maxabse,figSavePath+"graph_pack_error.fig");
pause(0.05);
close(h_mse_maxabse);

% Mostrar mensaje de finalización del programa
disp("Gráficos realizados");
end % function
```

Subfunciones

Código 4: Generación de datos de entrada mediante *camino aleatorio saturado*

```
function u = gen_satRandWalk(outSize,outRange,switchProb)

% Calcular tamaño del rango y crear rango expandido (+-20%)
rangeTot = outRange(2)-outRange(1);
expRange = [outRange(1)-rangeTot*0.2, outRange(2)+rangeTot*0.2];
expRangeTot = expRange(2)-expRange(1);

% Inicializar vector, y rellenar la primera muestra con un valor dentro
% del rango deseado
u = zeros(outSize);
u(:,1) = rand(outSize(1),1)*rangeTot + outRange(1);

% Obtener el resto de muestras dentro del rango expandido y saturarlas
% para que pertenezcan al rango deseado
for i=2:outSize(2)
    if rand <= switchProb
        val = rand(outSize(1),1)*expRangeTot + expRange(1); % Generar
        val = min(max(val,outRange(1)),outRange(2));        % Saturar
    else
        val = u(:,i-1);
    end % if
    u(:,i) = val;
end % bucle for i
end % function
```

Código 5: Entrenamiento de redes NARX

```
function varargout = NARX_trainNet(trainInput,trainTarget,numNeuron,...
                                                          inDelay,tgDelay)

%% Comprobar número de parámetros de salida
if (nargout < 1) || (nargout > 2)
    error("Número inválido de salidas: debe ser 1 o 2");
end

%% Crear y configurar red NARX
% Crear la red NARX
net = narxnet(inDelay,tgDelay,numNeuron,'open','trainlm');

% Configurar ratios de entrenamiento, validación y test de la red, y
% deshabilitar la ventana de entrenamiento
net.divideParam.trainRatio = 1;
net.divideParam.valRatio = 0;
net.divideParam.testRatio = 0;
net.trainParam.showWindow = false;

%% Formatear los datos de entrenamiento para poder ser usados
% Pasar los datos al formato empleado para trabajar con RNA
[inNet,~] = tonndata(trainInput,true,false);
[tgNet,~] = tonndata(trainTarget,true,false);

```

```
% Adecuar los datos a los retardos de la red
[inShift,iniInStates,iniLayStates,tgShift] = preparets(net,inNet,{},...
                                                       tgNet);

%% Entrenar la red
[net,trainRec] = train(net,inShift,tgShift,iniInStates,iniLayStates,...
    'useParallel','no','showResources','no','useGPU','no');

%% Asignar salidas
varargout{1} = net;
if nargout >= 2
    varargout{2} = trainRec;
end
end % function
```

Código 6: Validación en lazo abierto

```
function varargout = NARX_val_open(net,valInput,valTarget)

%% Comprobar número de parámetros de salida
if (nargout < 1) || (nargout > 3)
    error("Número inválido de salidas: debe ser entre 1 y 3");
end

%% Simplificar red
% Si se introduce una red "normal" se debe "reducir", ya que es el tipo
% que emplea la función NARX_Paso
if isa(net,"network")
    net = load_NN_data(net);
end

%% Calcular salida mediante red neuronal
outValNN = zeros(size(valTarget));
for k=(net.maxDelay+1):length(valInput)
    NNin = reshape(valInput(:,k-net.inPar.delay),[],1);
    NNtg = reshape(valTarget(:,k-net.tgPar.delay),[],1);
    outValNN(:,k) = NARX_Paso(net,NNin,NNtg);
end
outValNN = outValNN(:,net.maxDelay+1:end);

%% Obtener salidas
% Calcular el error absoluto cometido por la red
valTarget = valTarget(:,net.maxDelay+1:end);
e_abs = abs(outValNN-valTarget);

% Asignar salidas
varargout{1} = outValNN;
if nargout >= 2
    varargout{2} = mean(e_abs.^2,2);
    if nargout >= 3
        varargout{3} = max(e_abs,[],2);
    end
end
end % function
```

Código 7: Validación en lazo cerrado

```
function varargout = NARX_val_closed(net,valInput,valTarget)

%% Comprobar número de parámetros de salida
if (nargout < 1) || (nargout > 3)
    error("Número inválido de salidas: debe ser entre 1 y 3");
end

%% Simplificar red
% Si se introduce una red "normal" se debe "reducir", ya que es el tipo
% que emplea la función NARX_Paso
if isa(net,"network")
    net = load_NN_data(net);
end

%% Calcular salida mediante red neuronal
outValNN = zeros(size(valTarget));
outValNN(:,1:net.maxDelay) = valTarget(:,1:net.maxDelay);
for k=(net.maxDelay+1):length(valInput)
    NNin = reshape(valInput(:,k-net.inPar.delay),[],1);
    NNtg = reshape(outValNN(:,k-net.tgPar.delay),[],1);
    outValNN(:,k) = NARX_Paso(net,NNin,NNtg);
end
outValNN = outValNN(:,net.maxDelay+1:end);

%% Obtener salidas
% Calcular el error absoluto cometido por la red
valTarget = valTarget(:,net.maxDelay+1:end);
e_abs = abs(outValNN-valTarget);

% Asignar salidas
varargout{1} = outValNN;
if nargout >= 2
    varargout{2} = mean(e_abs.^2,2);
    if nargout >= 3
        varargout{3} = max(e_abs,[],2);
    end
end
end % function
```

Código 8: Función de reducción de red: generación de red reducida

```
function NN_reduc = load_NN_data(in)

% Comprueba si la entrada es un nombre de archivo o una red neuronal
if ischar(in) || isstring(in)
    net = importdata(in);
elseif isa(in,"network")
    net = in;
else
    error("Entrada inválida: debe ser un nombre de archivo ..." + ...
        "o una variable de tipo 'network'");
end

% Extrae la mínima información necesaria en la salida:
%  - Retardo máximo de la red
%  - Parámetros de normalización de input
%  - Retardos de input
%  - Número de input
%  - Parámetros de normalización de target
%  - Retardos de target
%  - Número de target
%  - Matrices de pesos y bias de la capa oculta
%  - Matrices de pesos y bias de la capa de salida
NN_reduc = struct('maxDelay',net.numInputDelays,...
    'inPar',struct('xmin',net.inputs{1}.processSettings{1}.xmin,...
                   'ymin',net.inputs{1}.processSettings{1}.ymin,...
                   'gain',net.inputs{1}.processSettings{1}.gain,...
                   'delay',net.inputWeights{1,1}.delays,...
                   'size',net.inputs{1}.size),...
    'tgPar',struct('xmin',net.inputs{2}.processSettings{1}.xmin,...
                   'ymin',net.inputs{2}.processSettings{1}.ymin,...
                   'gain',net.inputs{2}.processSettings{1}.gain,...
                   'delay',net.inputWeights{1,2}.delays,...
                   'size',net.inputs{2}.size),...
    'hidLay',struct('Wx',net.IW{1,1},...
                    'Wy',net.IW{1,2},...
                    'b',net.b{1}),...
    'outLay',struct('W',net.LW{2,1},...
                    'b',net.b{2}));
end % function
```

Código 9: Función de cálculo matricial de la salida de la red

```
function out = NARX_Paso(net,input,target)

% La función usa la estructura de red reducida.Si se ha pasado la
% red como red normal, se pasa a este formato.
if isa(net,"network")
    net = load_NN_data(net);
end

% Revisar que ambas entradas tienen el mismo número de muestras
% (columnas)
if size(input,2) == size(target,2)
    nSamples = size(input,2);
else
    error("Input y target deben tener el mismo número de muestras");
end

% Normalizar entradas
xIn = norm_AA(input,net.inPar.xmin,net.inPar.ymin,net.inPar.gain);
yIn = norm_AA(target,net.tgPar.xmin,net.tgPar.ymin,net.tgPar.gain);

% Pasar entradas por capa oculta y capa de salida
outHidden = tansig(net.hidLay.Wx*xIn+net.hidLay.Wy*yIn+...
                   repmat(net.hidLay.b,[1,nSamples]));
outNorm = net.outLay.W*outHidden+repmat(net.outLay.b,[1,nSamples]);

% Desnormalizar la salida
out = denorm_AA(outNorm,net.tgPar.xmin,net.tgPar.ymin,net.tgPar.gain);
end % function

%% Subfunciones: normalización y desnormalización
% Normalizar entradas. Las primeras líneas son para adaptar las
% distintas dimensiones de las matrices para que sean congruentes entre
% ellas (necesario cuando se emplea más de una muestra de input/target)
function inNorm = norm_AA(in,xmin,ymin,gain)
    nIn = size(in,1)/size(gain,1);
    xmin = repmat(xmin,[nIn,1]);
    gain = repmat(gain,[nIn,1]);
    inNorm = ymin+gain.*(in-xmin);
end

function out = denorm_AA(outNorm,xmin,ymin,gain)
    out = xmin+(outNorm-ymin)./gain;
end
```

Código 10: Sistema SNL5

```
function y = SNL5_sys(u_in)

% Pasar un vector entero de entradas por la función
u = zeros(1,length(u_in)+2);
u(3:end) = u_in;
y = zeros(size(u));
for k=2:(numel(y)-1)
    %y(k+1)= (1.5*y(k)*y(k-1))/(1+y(k).^2+y(k-1).^2)+
%           0.7*sin(0.5*(y(k)+y(k-1)))*cos(0.5*(y(k)+y(k-1)))+1.2*u(k);
    y(k+1) = (1.5*y(k)*y(k-1))/(1+y(k).^2+y(k-1).^2)+...
                                          0.35*sin(y(k)+y(k-1))+1.2*u(k);
end % bucle for k
y = y(3:end);
end % function
```

Código 11: Sistema SNL1

```
function y = SNL1_sys(u)

% Pasar un vector entero de entradas por la función
y = zeros(size(u));
y(1) = 0;
for k=1:(numel(y)-1)
    y(k+1) = u(k).^3 + y(k)/(1+y(k).^2);
end % bucle for k
end % function
```

Código 12: Ejemplo de script que integra todas las funciones desarrolladas

```
%% Integración del desarrollo de modelos neuronales: ejemplo SISO
clearvars;clc;close all;    % Limpiar espacio de trabajo

%% Inicialización de parámetros de barrido y datos de entrenamiento
% Parámetros de barrido
numNeur = 2:6;      % Número de neuronas a emplear en las redes
inDel = {0,0:1};    % Retardos de entradas a emplear
tgDel = {1,1:2};    % Retardos de target a emplear
numRep = 4;         % Número de redes calculadas para cada combinación
N = 300;            % Número de muestras de entrada y validación

% Entradas y targets de entrenamiento (iguales para todas las redes). Su
% formato debe ser que cada timestep se representa en una columna (por
% tanto, cada input/target se representa como una fila)
trData.input = gen_satRandWalk([1,N],[-2.5, 2.5],0.2);
trData.target = SNL5_sys(trData.input);

% Entradas y targets de validación (iguales para todas las redes, pero
% distintas a las de entrenamiento). Mismo formato que el anterior.
valData.input = gen_satRandWalk([1,N],[-2.5, 2.5],0.2);
valData.target = SNL5_sys(valData.input);

%% Realizar desarrollo de modelos neuronales
netData = NARX_paramSweep(numNeur,inDel,tgDel,numRep,trData,valData);
netData_proc = NARX_process_netData(netData);
NARX_plot_netData(netData_proc,valData,"TFM_figures\SISO\");
```

XVII Simposio CEA de Control Inteligente

27-29 de junio de 2022, León

Estudio de estructuras neuronales NARX para reproducir el comportamiento de sistemas con dinámicas complejas

Aimar Alonso, Asier Zabaljauregi, Eloy Irigoyen, Mikel Larrea

Universidad de País Vasco / Euskal Herriko Unibertsitatea (UPV/EHU)

To cite this article: Alonso, A., Zabaljauregi, A, Irigoyen, E., Larrea, M., 2022. iMO-NMPC strategy development: First steps for implementation on industrial hardware. XVII Simposio CEA de Control Inteligente, 1-5.

Resumen

Este trabajo presenta un estudio preliminar donde se valorará la eficiencia de las redes neuronales artificiales de topología NARX (Nonlinear Autoregressive eXogenous) en la reproducción del comportamiento de sistemas con dinámicas complejas. Estas estructuras neuronales se diseñarán para reproducir tanto sistemas monovariables, como multivariables, siguiendo un mismo planteamiento metodológico. Los mencionados estudios están dirigidos a proporcionar dichos modelos neuronales a futuras estrategias de control dependientes de modelos dinámicos, como es el caso del control predictivo no lineal basado en modelos, el cual constituye una línea de trabajo dentro del grupo de investigación de control intelgente (GICI) de la UPV/EHU.

Palabras clave: Red Neuronal, Sistemas no lineales, Identificación

Study of NARX neural structures to reproduce the behaviour of systems with complex dynamics.

Abstract

This work presents the methodology used by the Intelligent Control Research Group (GICI) at UPV/EHU, for the development of intelligent control strategies and their further implementation in real time platforms. In this way, it is intended to provide validation of such strategies not only in simulation level but in alternative industrial hardware. The presented case that is currently being developed is the iMO-NMPC strategy which integrates predictive control strategies, evolutionary algorithms for optimization and neural networks for system modelling. The employed methodology involves the simulation platform MATLAB/Simulink ®.

Keywords: Neural Networks, Nonlinear systems, *S-Function*, Identification

1. Introducción

En el mundo real, los sistemas a controlar son complejos, ya que incluyen no linealidades e interacción entre las diferentes entradas y salidas de este. Esto causa que la identificación de las características del sistema para desarrollar un modelo sea un proceso complicado, especialmente si el modelo será usado en métodos como *Model Predictive Control* (MPC), que requieren de una buena aproximación de los parámetros del proceso.

En este ámbito, se ha extendido el uso de Redes Neuronales Artificiales (RNA) para la modelización de estos sistemas, ya que se ha comprobado que son aproximadores universales (Hornik et al., 1990), (Jagannathan y Lewis, 1996), (Perrusquía y Yu, 2021). Estos modelos neuronales son así implementables en las estrategias de control mencionadas, como MPC o su versión no lineal (NMPC) (Camacho y Bordons, 2007), obteniendo buenos resultados (Bamimore et al., 2021).

En este trabajo, se busca modelizar sistemas no lineales mediante RNA, tanto con una única salida y entrada (*Single Input Single Output*, SISO), como con múltiples (*Multiple Input Multiple Output*, MIMO), comparando la respuesta obtenida respecto al modelo real.

2. Presentación del problema

Los sistemas empleados en este trabajo son sintéticos, que buscan valorar el rendimiento de este método de identificación, debido a que presentan regiones con grandes no linealidades. Estos sistemas han sido empleados anteriormente en otros trabajos relacionados con la identificación de modelos no lineales (Larrea et al., 2015). Para ofrecer una representación de las no linealidades de ambos sistemas, se presenta en la Figura 1 un

Correos electrónicos: aalonso198@ikasle.ehu.eus (Aimar Alonso), azabaljauregi001@ikasle.ehu.eus (Asier Zabaljauregi), eloy.irigoyen@ehu.eus (Eloy Irigoyen), m.larrea@ehu.eus (Mikel Larrea)

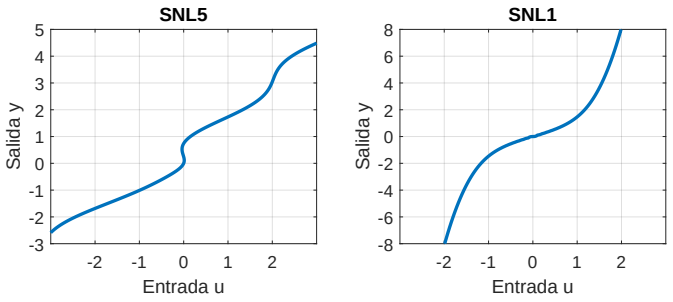

Figura 1: Relación de entradas/salidas de los sistemas en régimen estacionario

conjunto de puntos de operación en régimen estacionario, mediante la relación entrada/salida, cuyos rangos abarcan dinámicas no lineales suficientemente significativas.

Estos modelos son ambos modelos monovariables y se ha estudiado uno de ellos, el SNL5, para el primer caso (SISO). Para el estudio del segundo caso (MIMO), sin embargo, se ha generado un sistema multivariable no acoplado con ambos. Para ello se ha creado un sistema compuesto por SNL1 y SNL5.

Las RNA empleadas son de arquitectura NARX en lazo abierto, haciéndose un estudio del rendimiento de las mismas con distintos valores para el número de neuronas en la capa oculta y retardos de las entradas, con el objetivo de obtener redes neuronales que realicen una buena aproximación al sistema real, pero sin sobredimensionamiento.

2.1. *SISO*

En primer lugar, se ha estudiado el sistema no lineal denominado *SNL5*:

$$y_{k+1} = \frac{1{,}5 \cdot y_k \cdot y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + 0{,}7 \cdot \sin\left(0{,}5\left(y_k + y_{k-1}\right)\right) \cdot \cdot \cos\left(0{,}5\left(y_k + y_{k-1}\right)\right) + 1{,}2u_k \tag{1}$$

Al ser un sistema con una única entrada y una única salida, se variarán el valor del número de neuronas entre 2 y 6, siendo los retardos de las entradas externas (u) 0 o 0/1, y los retardos de la salida real (*target*) 1 o 1/2.

2.2. *MIMO*

En segundo lugar, se ha estudiado el caso de un sistema MIMO desacoplado, compuesto por el sistema *SNL5*, y otro sistema no lineal denominado *SNL1*:

$$\begin{cases} y_{1,k+1} = \dfrac{1{,}5 \cdot y_{1,k} \cdot y_{1,k-1}}{1 + y_{1,k}^2 + y_{1,k-1}^2} + 0{,}7 \cdot \sin\left(0{,}5\left(y_{1,k} + y_{1,k-1}\right)\right) \cdot \\ \qquad\qquad \cdot \cos\left(0{,}5\left(y_{1,k} + y_{1,k-1}\right)\right) + 1{,}2u_{1,k} \\ y_{2,k+1} = \dfrac{y_{2,k}}{1 + y_{2,k}^2} + u_{2,k}^3 \end{cases} \tag{2}$$

Al ser un sistema MIMO, más complejo, se aumentará el rango de valores de número de neuronas entre 8 y 20, en saltos de 2, para reducir el número de redes a entrenar; y los retardos, al igual que en el anterior caso, de las entradas externas (u) 0 o 0/1, y los retardos de la salida real (*target*) 1 o 1/2.

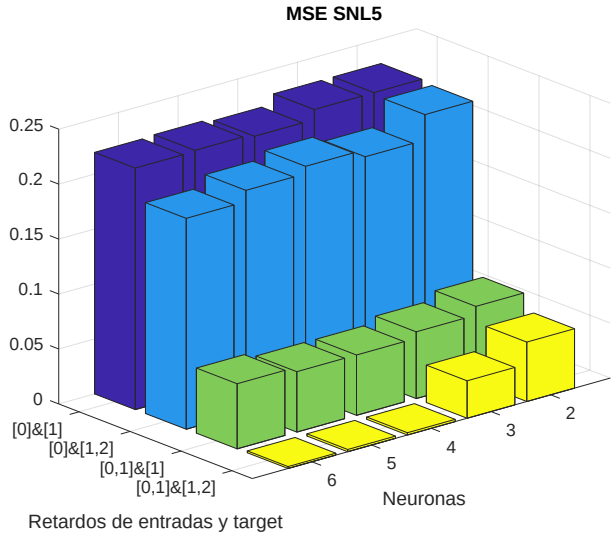

Figura 2: MSE de configuraciones de redes para el sistema SISO

3. Resultados

Para la obtención de los modelos neuronales se ha realizado un barrido de parámetros, siendo las entradas y *targets* de entrenamiento iguales para todas las redes.

Debido a que el punto de comienzo de cada entrenamiento es aleatorio, no se puede asegurar que la red sea óptima, por lo que se han entrenado 4 redes distintas para cada combinación de parámetros, y de estas se ha escogido aquella con mejor rendimiento en validación con entradas distintas a las de entrenamiento.

Tras obtener la mejor red para cada combinación de parámetros, se ha realizado un gráfico de barras del Error Cuadrático Medio (*Mean Square Error*, MSE) para tomar la decisión de la mejor red del conjunto. Finalmente, se ha realizado un gráfico de la validación de esta red para observar el comportamiento y la aproximación del sistema a identificar.

3.1. *SISO*

Realizando el proceso de entrenamiento de las redes mencionado, se ha obtenido el gráfico de MSE de las redes (figura 2). En este gráfico se observa que el parámetro que mayor impacto tiene en la disminución del error son los retardos de entrada y *target*. Así, observando el gráfico, se ha decidido elegir la red con 4 neuronas, y retardos de entradas 0/1 y retardos de *targets* 1/2. No se ha seleccionado un número mayor de neuronas, debido a que la reducción del error obtenida es muy pequeña en comparación con el aumento del tiempo de computación necesario.

Graficando la respuesta del sistema frente a una entrada u en el rango $[-2, 2]$ (figura 3), se ha obtenido una respuesta muy buena, siguiendo la salida real del sistema con error muy reducido.

3.2. *MIMO*

Tras realizar el entrenamiento de las redes con la combinación de parámetros mencionada, se han obtenido los gráficos de los MSE para ambas salidas (figuras 4 y 5). En estos gráficos,

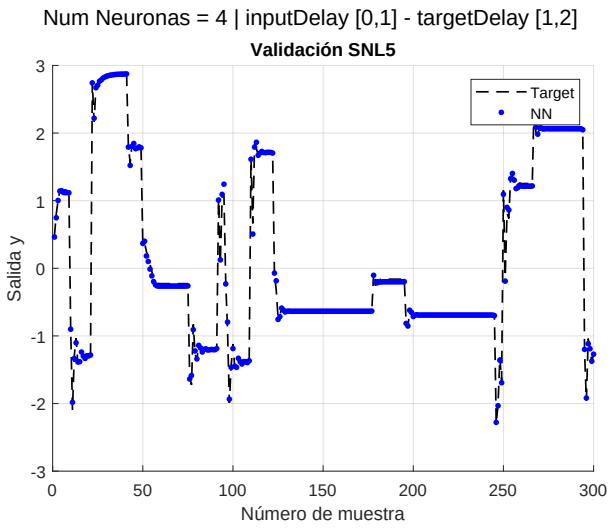

Figura 3: Respuesta de la red neuronal obtenida frente a una entrada de validación

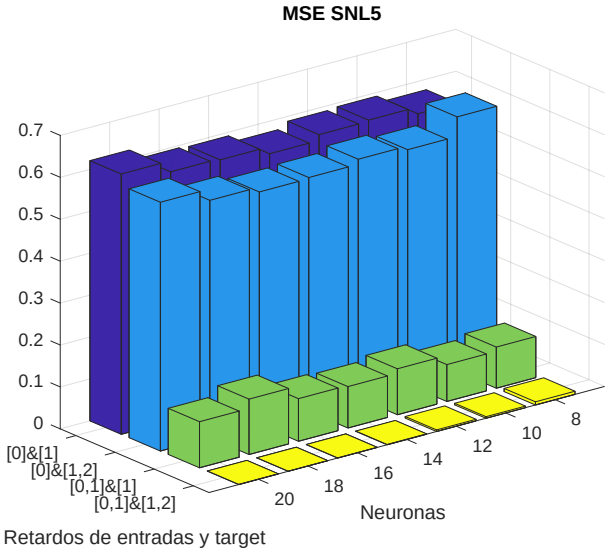

Figura 4: MSE de configuraciones de redes para el sistema MIMO: Subsistema SNL5

se puede observar de nuevo que el parámetro que mayor impacto tiene en la reducción del error es el retardo, en este caso, de las entradas *u*, aunque el retardo de los *target* también tiene impacto al combinarlo con el anterior.

Observando los errores, se ha decidido elegir la red neuronal con 14 neuronas, y retardos de entradas 0/1 y retardos de *targets* 1/2. De nuevo, no se ha seleccionado un número mayor de neuronas debido a la pequeña disminución del error relacionada con aumentar más el número de neuronas.

Realizando una validación con entradas en los rangos [-2,5, 2,5] para la primera entrada (*SNL5*), y [-1,8, 1,8] para la segunda entrada (*SNL1*) (figura 6), se comprueba que la respuesta para ambas salidas es excelente, y sigue muy de cerca ambas salidas simultáneamente, con un error muy ajustado.

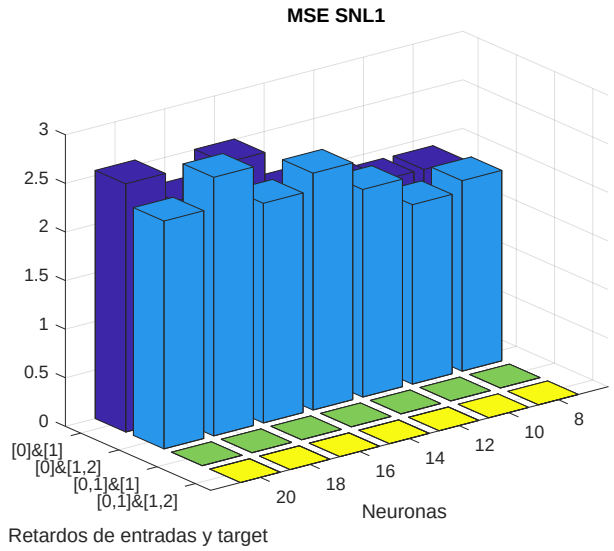

Figura 5: MSE de configuraciones de redes para el sistema MIMO: Subsistema SNL1

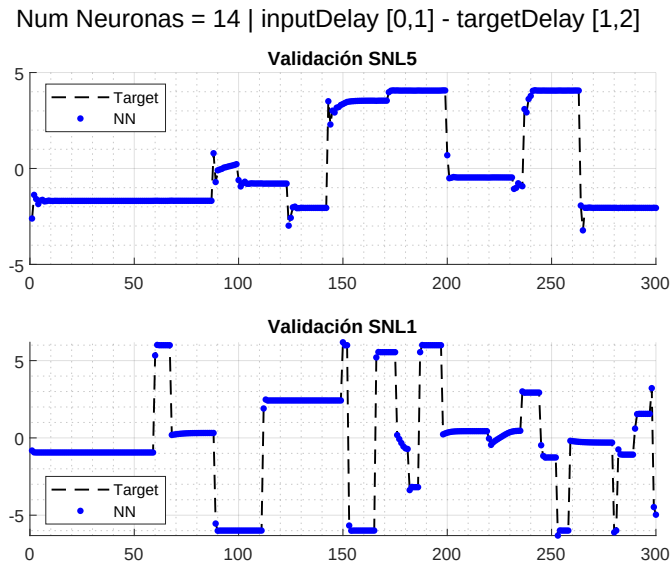

Figura 6: Respuesta ante validación de la red neuronal elegida para el sistema MIMO

4. Líneas de trabajo futuras

En el futuro, el objetivo es el desarrollo de un modelo neuronal MIMO de un sistema acoplado. Un ejemplo de este es el modelo *Twin-Rotor*, provisto por *Feedback Instruments Ltd* (Feedback Instruments, 2001); que se trata de un modelo MIMO 2x2 fuertemente acoplado.

Adicionalmente, también queda ampliar el uso de estos modelos neuronales a estrategia de Control Inteligente, como el iMO-NMPC (Valera García et al., 2012) para estos sistemas acoplados, en los que resulta crucial que el error del modelo sea mínimo para que el control funcione.

Otro posible desarrollo futuro de estos modelos neuronales pasa por la inclusión de otros parámetros no contemplados, como la adición de capas adicionales, o otras arquitecturas de redes.

Finalmente se buscará la integración de los modelos obtenidos en aplicaciones de tiempo real de MATLAB/Simulink a

modo de código adicional de s-functions.

Agradecimientos

Este trabajo se ha desarrollado en el marco del proyecto PID2020-120087GB-C22 financiado por el Ministerio de Ciencia e Innovación del Gobierno de España.
(AEI / http://dx.doi.org/10.13039/501100011033)

Referencias

Bamimore, A., Osinuga, A., Kehinde-Abajo, T., Osunleke, A., Taiwo, O., 2021. A comparison of two artificial neural networks for modelling and predictive control of a cascaded three-tank system. IFAC-PapersOnLine 54 (21), 145–150, control Conference Africa CCA 2021.
DOI: https://doi.org/10.1016/j.ifacol.2021.12.025

Camacho, E. F., Bordons, C., 2007. Nonlinear Model Predictive Control: An Introductory Review. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–16.
DOI: 10.1007/978-3-540-72699-9$_1$

Feedback Instruments, L., 2001. Twin Rotor MIMO System Control Experiments 33-949S. Feedback.

Hornik, K., Stinchcombe, M., White, H., 1990. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural Networks 3 (5), 551–560.
DOI: https://doi.org/10.1016/0893-6080(90)90005-6

Jagannathan, S., Lewis, F., 1996. Identification of nonlinear dynamical systems using multilayered neural networks. Automatica 32 (12), 1707–1712.
DOI: https://doi.org/10.1016/S0005-1098(96)80007-0

Larrea, M., Larzabal, E., Irigoyen, E., Valera, J., Dendaluce, M., 2015. Implementation and testing of a soft computing based model predictive control on an industrial controller. Journal of Applied Logic 13 (2, Part A), 114–125, sI: SOCO12.
DOI: https://doi.org/10.1016/j.jal.2014.11.005

Perrusquía, A., Yu, W., 2021. Identification and optimal control of nonlinear systems using recurrent neural networks and reinforcement learning: An overview. Neurocomputing 438, 145–154.
DOI: https://doi.org/10.1016/j.neucom.2021.01.096

Valera García, J. J., Gómez Garay, V., Irigoyen Gordo, E., Artaza Fano, F., Larrea Sukia, M., 2012. Intelligent multi-objective nonlinear model predictive control (imo-nmpc): Towards the 'on-line' optimization of highly complex control problems. Expert Systems with Applications 39 (7), 6527–6540.
DOI: https://doi.org/10.1016/j.eswa.2011.12.052