UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

GRADO EN INGENIERÍA MECÁNICA

TRABAJO FIN DE GRADO

"DESARROLLO DE UN MODELO DINÁMICO MEDIANTE REDES NEURONALES RECURRENTES PARA PREDECIR EL COMPORTAMIENTO DE LA COMBUSTIÓN EN NUEVOS MOTORES MCIA"

AUTOR:

GUILLEM PÉREZ GARCÍA

TUTOR:

PAU BARES MORENO

CURSO ACADÉMICO: 2020-2021

Índice General

Índice de figuras

Índice de tablas

Tabla 1. Identificación de parámetros

RESUMEN

El presente trabajo tiene como objetivo estudiar el comportamiento de la combustión de nuevos motores MCIA en condiciones estáticas y en transitorios acusados donde las dinámicas del ciclo de aire no pueden ser despreciadas. Variables como el gasto de aire, las presiones en cámara, o medidas de oxígeno serán estudiadas para predecir el comportamiento del motor. Para el estudio del sistema estático se han utilizado redes feed-forward y para modelar el sistema dinámico se han utilizado un tipo de redes neuronales recurrentes, llamadas NARX. En ambos modelos se han hecho un estudio con redes basadas, por un lado en Extreme Learning Machine (ELM) y por otro lado en Backpropagation. El primer método permite una predicción aceptable con un coste computacional mínimo mientras que el segundo implica un mayor coste de calibración y cálculo para adaptarse mejor a la solución esperada.

1. INTRODUCCIÓN

Permitir a los ordenadores modelar el mundo que nos rodea lo suficientemente bien como para que se hable de inteligencia ha sido el foco de más de medio siglo de investigaciones. Para conseguirlo, sería necesario almacenar una gran cantidad de información en ellos. Parece impensable que la acción de gestionar toda esta información se realice manualmente, así que con el objetivo de que los ordenadores puedan usarla para contestar algunas preguntas y generalizar sobre nuevos contextos, muchos investigadores han recurrido a los algoritmos de aprendizaje para capturar una enorme fracción de toda esta información [1].

El mecanismo necesario para conseguir que un ordenador tenga la capacidad de resolver un problema se llama algoritmo, que no es más que una secuencia de instrucciones que tienen como objetivo la transformación de unos parámetros de entrada para obtener una salida. Existen gran cantidad de algoritmos, por lo que es interesante siempre ir en busca del más eficiente.

En los últimos años, el avance de la tecnología ha permitido que el almacenamiento y procesado de gran cantidad de datos sea posible y sencillo. Por ello, lo realmente importante en estos momentos es saber, una vez están almacenado, como analizarlos, para que así toda esa información pueda ser útil y se pueda utilizar, por ejemplo, para predecir valores futuros de una función.

En este punto es donde entra el *machine learning* (aprendizaje automático), el cual es una rama de inteligencia artificial dentro del campo de la ciencia de computación. Este tiene como objetivo que los ordenadores sean capaces de resolver problemas de manera autónoma (que puedan aprender). Estas técnicas desarrolladas se basan en que, a raíz de una serie de variables y tras aplicársele un algoritmo de aprendizaje se conviertan en la variable de salida deseada. En este proceso, aunque no exista una correlación aparente entre las diferentes entradas y las salidas que se tienen como objetivo, se puede conseguir una útil aproximación, por lo que, aun no conociendo cuales son las conexiones existentes entre estos datos, se puede conseguir extraer patrones de ellos, lo cual ayuda a la hora de analizarlos.

Dentro de un campo tan amplio como el del *machine learning,* el desarrollo del presente proyecto se va a centrar en el sector de la redes neuronales artificiales. Este tipo de técnicas están basadas en el comportamiento del sistema nervioso humano, un sistema conformado por neuronas que interaccionan entre sí para producir una salida concreta según con lo que se haya estimulado al inicio.

En el modelado de motores de combustión interna alternativos (cuyas siglas son MCIA) se trabaja con una gran cantidad de datos y variables distintas, por lo que extraer información relevante de los archivos de almacenamiento (en los cuales hay miles de valores) es un trabajo complicado. El procesado de estos datos es casi imposible para el ser humano, por lo que en el momento en el que se tuvo que abordar este tipo de

problemas, se empezó a investigar en el campo de las redes neuronales artificiales para solucionarlos.

Con este tipo de redes se pueden conseguir relaciones entre diferentes parámetros de un motor, como pueden ser la presión en cámara o el gasto de aire. Estos patrones que surgen al aplicar estas técnicas pueden ayudar a entender el proceso interno del motor, y a realizar predicciones siempre y cuando se parta de la premisa de que, bajo unas condiciones de trabajo determinadas, el futuro de ese motor sea similar al momento en el que se tomaron los datos utilizados.
Esto quiere decir que, conociendo el comportamiento de la combustión de estos motores bajo unas condiciones de carga, velocidad, gasto de fuel, etc. se puede conocer cómo se comportará en momentos posteriores a la finalización de la toma de datos.

2. REDES NEURONALES ARTIFIALES (RNAs)

Las redes neuronales artificiales son modelos computacionales creados para enfrentarse de manera efectiva a problemas con gran cantidad de datos y complejos. Como bien indica su nombre, estos modelos tienen una forma de procesamiento de datos que intenta simular a la que ocurre en las redes biológicas del cerebro humano.

El cerebro humano funciona de manera completamente distinta al funcionamiento de un ordenador digital convencional. Este consiste en un sistema altamente complejo, no lineal y paralelo. Esto quiere decir que puede realizar muchas operaciones de manera simultánea, a diferencia de los ordenadores comunes que son de tipo secuencial (realizan únicamente una operación a la vez) [2].

Estas redes, al igual que la del cerebro, están compuestas de muchos unidades elementales llamadas neuronas. Las cuales de por sí no conseguirían una transmisión de información, pero el conjunto genera una red de transmisión muy eficiente.

Las características principales de estos modelos son:

- Aprendizaje: adquirir conocimientos a través de la experiencia y el entrenamiento.
- Adaptación: las RNAs pueden cambiar su manera de trabajar en función del entorno de los datos de entradas.
- Tolerancia a fallos: poseen una gran capacidad de absorber fallos sin perjudicar el modelado.
- Generalización: por el funcionamiento de estos modelos, pueden obtener resultados satisfactorios a pesar de las fluctuaciones de las entradas, siempre dentro de un margen.
- Comportamiento no-lineal: pueden procesar información que no tenga una relación directa aparente.

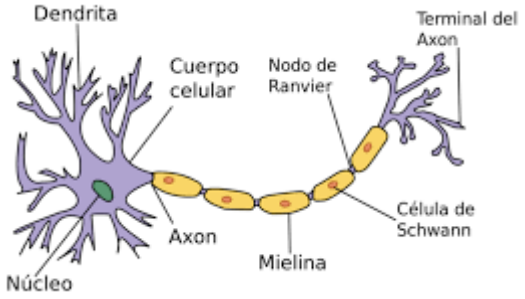

Figura 1. Partes de una neurona

Para entender el funcionamiento de estos modelos es necesario primero conocer, cómo funciona el sistema neuronal del cerebro. Una neurona posee el aspecto que se

aprecia en la figura 1. Consta de dendritas, de un soma celular, de un axón y sus terminaciones.

Las neuronas, por lo general, son elementos que en solitario tratan la información de manera lenta. Sin embargo, el cerebro compensa de una manera extraordinaria esa lentitud, generando una red con gran cantidad de ellas, las cuales están todas interconectadas. Esto conlleva que el cerebro sea un sistema altamente eficiente, siendo esta eficiencia aproximadamente de 10^{16} operaciones/s, lo cual es del orden de 10^{10} veces mayor que la de los mejores ordenadores de la actualidad [2].

Las neuronas codifican la información de salida en unos impulsos llamados *potenciales de acción,* que se generan en el soma celular, estos impulsos se propagan a través del axón hasta llegar a la sinapsis, desde donde continua hasta las dendritas de la neurona siguiente.

Una sinapsis es una interconexión entre dos neuronas. Un dibujo esquemático de ella se incluye en la figura 2. En ella, el botón sináptico corresponde al término del axón de una neurona presináptica, y la dendrita es la correspondiente a una neurona postsináptica.

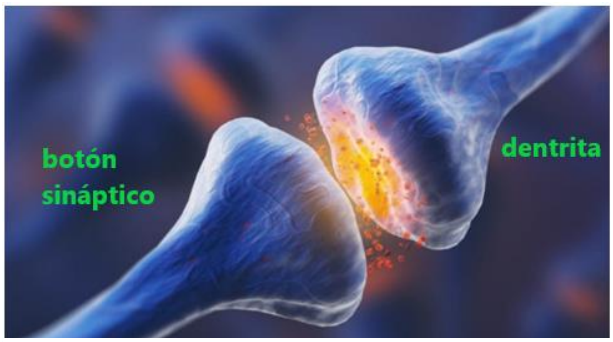

Figura 2. Sinapsis

La característica más importante para el devenir de este trabajo es que en el soma celular se suman las entradas de todas las dendritas. Si estas entradas sobrepasan un cierto umbral, entonces se transmitirá un pulso a lo largo del axón, en caso contrario no transmitirá. Esto es en lo que se basan las redes neuronales artificiales y por lo tanto es la premisa con la que trabajaremos.

2.1 FUNCIONAMIENTO

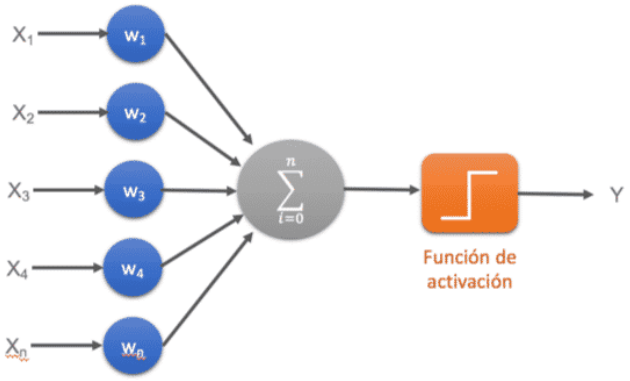

Figura 3. Red neuronal simple

El funcionamiento de una RNA, en su estado más elemental (perceptrón de una capa y una neurona), se basa en una serie de datos de entrada, llamados Entradas (X_i), las cuales equivaldrían a las dendritas, los cuales son multiplicados por los pesos sinápticos (W_i), que son los elementos de conexión entre la entrada y las neuronas. Los resultados de esta multiplicación que llegan a la neurona son sumados, convirtiéndose así en la entrada a la capa oculta (*hidden layer)*. Este es el proceso que ocurre en el soma celular.

$$z_i = \sum_{i=1}^{n} \omega_{ij} * x_j \quad (1)$$

Una vez aquí, se aplica una función no lineal llamada función de activación. La cual tiene un cierto valor umbral θ, que tiene que ser sobrepasado por la neurona para activarse. Las funciones de activación más usadas son:

- Escalón binario:

$$g(x) = \begin{cases} 0 \; si \; x \leq 0 \\ 1 \; si \; x > 0 \end{cases}$$

Figura 4. Función escalón binario

- Función Sigmoide:

$$g(x) = \frac{1}{1 + e^{-x}}$$

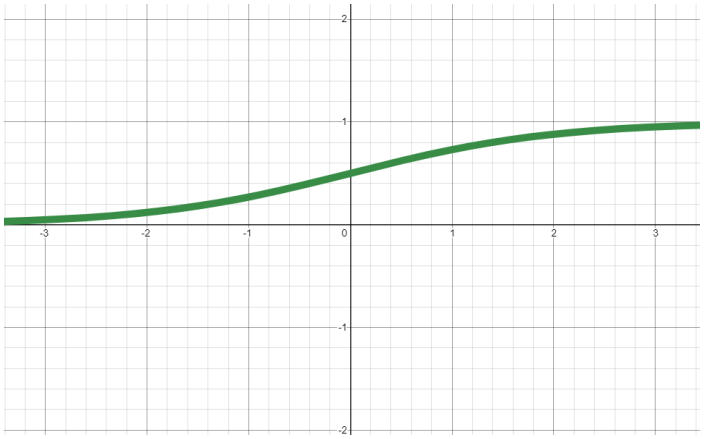

Figura 5. Función sigmoide

- Función Tangente Hiperbólica:

$$g(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

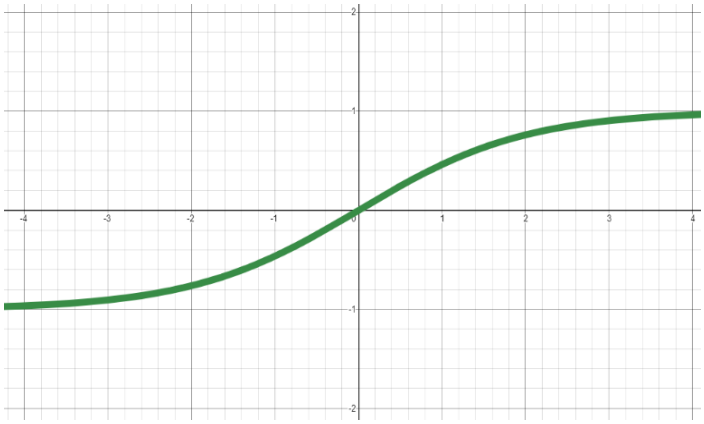

Figura 6. Función Tangente hiperbólica

Al finalizar esta operación, la información sale de la neurona. Esto sería la salida de la RNA y bien podría ser usada como resultado para el estudio o para continuar un proceso de entramiento en neuronas siguientes.

Entre los tipos de RNAs podemos destacar las redes neuronales prealimentadas (más conocidas como feed-forward), las cuales se basan un tratamiento unidireccional (hacia delante) de la información y las redes neuronales recurrentes, la cuales generan bucles de retroalimentación entre las capas.

2.2. REDES NEURONALES FEED-FORWARD

Las redes neuronales feed-forward, como se ha comentado anteriormente, son un tipo de RNAs en el que la información tiene un funcionamiento lineal. Los datos de entrada fluyen hacia delante a través de las diferentes capas ocultas para desembocar en la capa de salida. Estas fueron las primeras en ser teorizadas, ya que su funcionamiento

es el más sencillo de entender. Como se ha explicado previamente, el tipo más sencillo dentro de esta categoría es el perceptrón de una capa, pero este tiene el inconveniente de que solo puede resolver problemas linealmente separables.

Esto significa que, en un supuesto en el que se tenga dos set de datos, estos puedan ser separados por un hiperplano, lo que haga que ambos no se entremezclen. Como se puede observar en la figura 6.

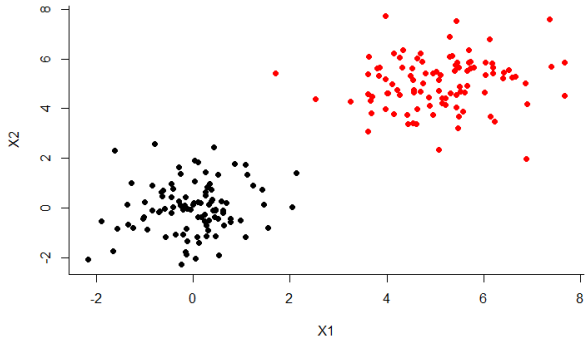

Figura 7. Datos linealmente separables

Para remediar este problema se creó una evolución de este modelo, el perceptrón multicapa.

2.2.1 Perceptrón multicapa (MLP)

Este modelo consiste en una agrupación de perceptrones simples con el objetivo de ser capaz de resolver problemas más complejos. Se compone de un conjunto de entradas, las cuales conforman la *capa de entrada.* Tras ella se pueden encontrar una o varias capas ocultas, donde están las neuronas. Al pasar por todas ellas se llega a la capa de salida, de la que sale el resultado que se quiere estudiar.
En estas capas ocultas se realizan los procedimientos que se han explicado con anterioridad, como el sumatorio de los pesos y la posterior aplicación de la función de activación.

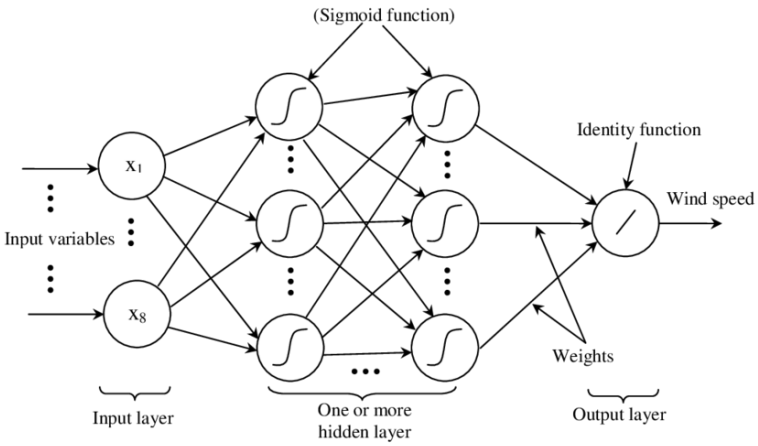

Figura 8. Perceptrón multicapa

El número de entradas y salidas viene marcado por el problema que se quiere resolver, pero estos modelos otorgan gran cantidad de flexibilidad al usuario encargado de encontrar una solución, ya que permiten manipular la cantidad de neuronas y el número de capas ocultas, para así encontrar el modelo más adecuado a cada tipo de problema. Esto da mucha versatilidad, pero puede conllevar algunos inconvenientes, como el aumento del coste computacional, porque puede llevar a tiempos de cálculo excesivos.

También puede haber de problemas de sobreajuste, en el que si se utilizan demasiados datos puede desembocar en una pérdida de generalización, en la cual los datos de salida se ajusten específicamente a los datos estudiados, pero fuera de ellos generen un error importante.

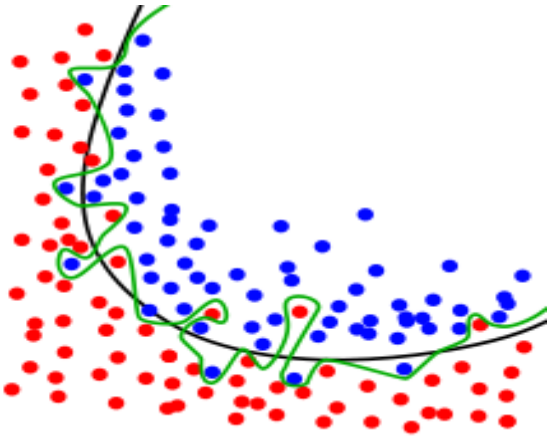

Figura 9. Sobreajuste

2.2.1.1 Algoritmo de retropropagación

La retropropagación, más conocida por su nombre en inglés backpropagation (abreviado por las siglas BP) es un conocido algoritmo de entrenamiento supervisado de redes neuronales.

En primer lugar, es interesante explicar el significado de entrenamiento supervisado. Este es un conjunto de técnicas que permite realizar predicciones futuras basándose en el comportamiento de los datos que se tienen antes de empezar a trabajar. Esto quiere decir que, a partir de una serie de datos de entrada y salida, se desea encontrar algún tipo de regla que permita aproximar, con el menor error posible, la respuesta para cualquier tipo de datos de entrada que se presenten.

Estas técnicas se utilizan principalmente problemas de regresión y clasificación, pero en el contexto de este trabajo se va a utilizar exclusivamente en problemas de regresión.

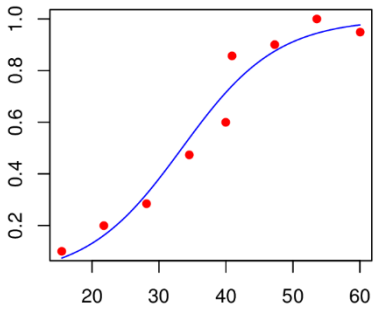

Figura 10. Regresión no-lineal

Retomando la explicación del algoritmo de retropropagación, este se encarga de evaluar la contribución al error general de cada neurona después de que un conjunto de datos sea procesado [3]. El objetivo principal de la retropropagación es modificar los pesos que unen las diferentes capas de la red, para entrenarla y que consiga convertir unas entradas arbitrarias en salidas con el menor error posible. Esto lo consigue calculando el gradiente de la función de coste (función que mide el error de predicción del modelo) con respecto a cada peso de la red. Este proceso lo realiza tras obtener el valor de salida, y en dirección contraria a la del flujo de la red, de ahí el nombre de retropropagación. Esta minimización del error se consigue utilizando el método de descenso del gradiente.

Este método de descenso el gradiente se utiliza para hallar esos valores mínimos las derivadas parciales de la función de coste con respecto a cada peso de la red. Esto se debe a que la derivada indica el valor y el sentido en el que se encuentra el mínimo más próximo.

El proceso para llevar a cabo con éxito la implementación de este algoritmo, suponiendo que se tiene una única capa oculta, es el siguiente:

1º) Calcular las salidas y_i con el proceso explicado anteriormente y con unos pesos aleatorios.

2º) Calcular el error total de la red:

$$E_{tot} = \sum_{i=1}^{m} \frac{1}{2}(y_i - salO_i)^2 \quad (2)$$

siendo *m* el número total de salidas, y_i el valor real del parámetro y $salO_i$el valor calculado por la red (la salida de la capa de salida).

3º) Calcular como afecta un cambio en los pesos que unen la capa oculta con la capa de salida (β_i) al error total. Esto quiere decir calcular la derivada parcial del error con respecto a β_i. Esta derivada se consigue aplicando la regla de la cadena, quedando la expresión que se observa en la ecuación (x).

$$\frac{\partial E_{tot}}{\partial \beta_i} = \frac{\partial E_{tot}}{\partial salO_i} * \frac{\partial salO_i}{\partial entO_i} * \frac{\partial entO_i}{\partial \beta_i} \quad (3)$$

siendo $entO_i$ la entrada a la capa de salida.

En este punto es interesante explicar cómo se obtiene la $\frac{\partial salO_i}{\partial entO_i}$, que no es tan intuitiva como el resto de las derivadas parciales. En el caso de que se use la función sigmoide como función de activación (la cual se va a utilizar a lo largo de todo el trabajo), siendo esta: $salO_i = \frac{1}{1+e^{-entO_i}}$, la derivada parcial con respecto a la entrada a la capa de salida viene dada por la ecuación (4).

$$\frac{\partial salO_i}{\partial entO_i} = salO_i * (1 - salO_i) \quad (4)$$

4º) Corregir los pesos β_i para disminuir el error.

$$\beta_i{}^{+} = \beta_i - \eta * \frac{\partial E_{tot}}{\partial \beta_i} \quad (5)$$

siendo $\beta_i{}^{+}$ el peso corregido y η un índice de entrenamiento, el cual es elegido por el usuario.

Hasta este paso se han calculado los pesos de la capa de salida, pero para calcular el resto de los pesos se tienen que utilizar los pesos β_i sin corregir. Cuando se acabe todo el proceso será el momento de actualizar todos los pesos a la vez.

5º) Calcular como afecta un cambio en los pesos que unen la capa de entrada con la capa oculta (ω_i) al error total.

$$\frac{\partial E_{tot}}{\partial \omega_i} = \frac{\partial E_{tot}}{\partial salH_i} * \frac{\partial salH_i}{\partial entH_i} * \frac{\partial entH_i}{\partial \omega_i} \quad (6)$$

siendo $salH_i$ la salida y $entH_i$ la entrada a la capa oculta.

Hay que tener en que $salH_i$ afecta a todas salidas de capa de salida, por lo que para calcular $\frac{\partial E_{tot}}{\partial salH_i}$ hay que seguir la siguiente expresión:

$$\frac{\partial E_{tot}}{\partial salH_i} = \sum_{j=1}^{m} \frac{\partial E_{sal\ j}}{\partial salH_i} \quad (7)$$

siendo $E_{sal\ j}$ el error de cada una de las salidas de red.

A su vez esta derivada parcial se calcula de la siguiente manera:

$$\frac{\partial E_{sal\ j}}{\partial salH_i} = \frac{\partial E_{sal\ j}}{\partial entO_i} * \frac{\partial entO_i}{\partial salH_i} \quad (8)$$

pudiendo calcular la primera parte de la multiplicación con datos de pasos anteriores.

Corregir los pesos ω_i para disminuir el error.

$$\omega_i{}^{+} = \omega_i - \eta * \frac{\partial E_{tot}}{\partial \omega_i} \quad (9)$$

siendo $\omega_i{}^{+}$ el peso corregido.

En esta explicación solo se tiene una capa oculta, pero si se necesitaran un número mayor de estas, el proceso sería el mismo, aunque más extenso en función de la cantidad que se requirieran.

El procedimiento completo se puede repetir, mediante un proceso iterativo, las veces que se considere oportuno para ir reduciendo el error hasta el valor deseado. Cuando se tienen los pesos actualizados se vuelve a calcular las salidas de la capa de salida y se vuelve a calcular el error. Si este no cumple con las expectativas se vuelve a aplicar el algoritmo y se vuelven a actualizar los pesos y así sucesivamente hasta lograr el objetivo.

El mayor inconveniente de este algoritmo es el tiempo de cálculo, ya que al tener que iterar hasta obtener el error deseado, si se quiere ser preciso, es necesario emplear mucho tiempo en el proceso de entrenamiento.

2.3 EXTREME MACHINE LEARNING (ELM)

Las redes neuronales, como se ha expuesto anteriormente, aumentan su complejidad cuando se van añadiendo más neuronas y más capas al modelo. Esto hace que el coste computacional aumente en gran medida, requiriendo ordenadores muy potentes y largos tiempos de espera para llegar a una solución. Por eso mismo, en los últimos tiempos se ha empezado a profundizar en las Máquinas de Aprendizaje Extremo o *Extreme Machine Learning (ELM)*. Esta técnica nació en la primera década del siglo XXI, de la mano del doctor Guang-Bin Huang, de la Nanyang Technological University de Singapur, el cual buscaba una solución para este problema.

El procesamiento de estas redes presenta una baja complejidad computacional, lo que acelera los tiempos de procesado creando un modelo rápido, pero sin sacrificar la capacidad de generalización.

La esencia de las ELM es que los parámetros de entrada a la capa oculta son generados de manera aleatoria, sin necesidad de ser entrenados, por lo que la carga computacional se reduce a la resolución de un sistema lineal. Esto hace que el diseño de ELM sea atractivo debido a su alta velocidad de entrenamiento y su buena capacidad de generalización [4].

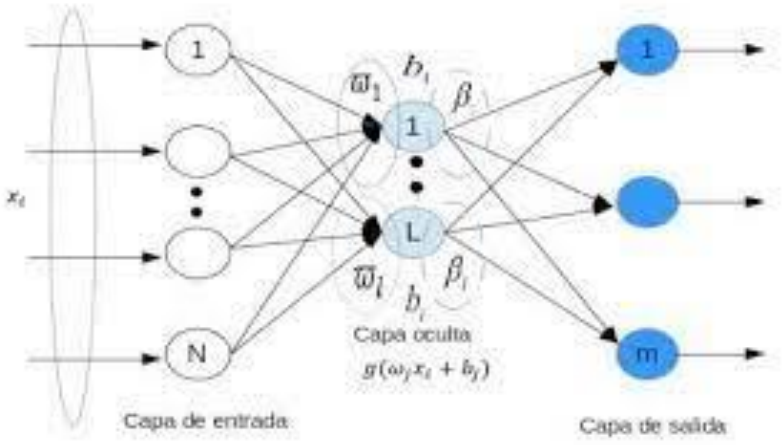

Figura 11. Estructura de una ELM

Las redes ELM tienen una arquitectura muy similar a la del Perceptrón anteriormente explicado, la diferencia reside en la manera en la que se entrena la red. Los pesos de entrada a la capa oculta (ω_i) y los biases (b_L) son generados de manera aleatoria.

Es interesante conocer el significado de bias, ya que puede ser un término con el que el público no está familiarizado. El bias es un tipo de error estadístico que mide la diferencia entre el valor esperado por el sistema y el valor real. Cuando se tiene un bias alto significa que el sistema es muy simple y no se ajusta a los datos de

entrenamiento. Para que un sistema esté bien entrenado el bias tiene que ser relativamente bajo (aunque sin que esto conlleve un sobre entrenamiento).

Tener estos datos de manera aleatoria hace que el objetivo principal de estos modelos sea encontrar el valor de los pesos (β_l) que conectan la capa oculta y la de salida.

Partiendo de set de datos con *R* muestras, de unas entradas (X_i, *i=1, 2, ..., n*), de unas salidas (Y_i, *i=1, 2, ..., m*) y un número *L* de neuronas. La forma de trabajar con estas redes consiste en:

1º) Calcular, primero, la entrada a la capa oculta.

$$z = \sum_{n=1}^{n} \omega_{ij} * X_n + b_L \quad (10)$$

Esto genera el sumatorio de todas las conexiones que entran a una única neurona. Esto hay que repetirlo para cada neurona de la red y para cada set de datos. Al final de este paso se debería obtener una matriz Z de dimensiones R x L.

2º) Calcular la salida de la capa oculta.

$$h_i = G(\omega_i, b_i, X) \quad (11)$$

Siendo G que la función de activación. Esto no es más que aplicar la función de activación G, que normalmente va a ser la sigmoide, a todos los valores de la matriz Z, tal que así: $G = \frac{1}{1+e^{-z}}$. Por lo tanto, se va a generar una matriz H con las mismas dimensiones que la anterior.

3º) Calcular los pesos β a través de la capa de salida.

Partiendo de la premisa de que:

$$f_L(x) = \sum_{i=1}^{L} \beta_i * h_i = y \quad (12)$$

β se resuelve minimizando el error de aproximación en el error cuadrático:

$$min_\beta \left|\left|\mathrm{y} - \mathrm{H}\right|\right| \quad (13)$$

Siendo ||* || la norma Frobenius, definida como la raíz cuadrada de la suma de los cuadrados absolutos de los elementos de una matriz.

La matriz H tiene la siguiente forma:

$$H = \begin{pmatrix} h(x_1) \\ \vdots \\ h(x_R) \end{pmatrix} = \begin{pmatrix} h_1(x_1) & \cdots & h_L(x_1) \\ \vdots & \ddots & \vdots \\ h_1(x_R) & \cdots & h_L(x_R) \end{pmatrix} \quad (14)$$

La matriz objetivo y tiene la siguiente forma:

$$y = \begin{pmatrix} {y_1}^T \\ \vdots \\ {y_R}^T \end{pmatrix} = \begin{pmatrix} y_{11} & \cdots & y_{1m} \\ \vdots & \ddots & \vdots \\ y_{R1} & \cdots & y_{Rm} \end{pmatrix} \quad (15)$$

Como resultado, la solución óptima para la ecuación (13) viene dada por:

$$\beta = H^{\gamma} * y \quad (16)$$

donde H^{γ} hace referencia a la inversa generalizada Moore-Penrose de la matriz de salida de la capa oculta H. La inversa generalizada Moore-Penrose puede ser calculada usando $(H^T * H)^{-1} * H^T$. Esto se puede aplicar cuando la matriz resultante de la operación $(H^T * H)$ es no singular, siendo esto que cuenta con una matriz inversa y cumple que $(H^T * H) * (H^T * H)^{-1} = I$. Como el número de datos empleados en el entrenamiento es normalmente superior al número de neuronas, tal que R>L, la ecuación (16) puede escribirse como:

$$\beta = (H^T * H)^{-1} * H^T * y \quad (17)$$

siendo H^T la transpuesta de la matriz H [3].

2.3.1 Extreme Machine Learning Regularizado (RELM)

Los modelos ELM tienden a tener problemas de sobreajuste, un fenómeno explicado con anterioridad en este mismo trabajo. Esto afecta a la habilidad de generalización del modelo, por lo que los métodos de regularización suelen ser útiles para mejorar en este aspecto y resolver el problema en este tipo de redes. Esto se consigue imponiendo una restricciones suaves al entrenamiento a través de un parámetro C, de valor positivo, llamado parámetro de regularización.

La expresión para obtener β en este tipo de modelos es la siguiente:

$$\beta = (H^T * H + \frac{I}{C})^{-1} * H^T * y \quad (18)$$

donde I es la matriz identidad. El parámetro C es el que marca la capacidad del ELM en lo que respecta a la generalización. La manera de conseguir el valor óptimo de este parámetro C suele ser prueba y error, teniendo que repetir este proceso y los diferentes sistemas sean realizados.

2.4 REDES NEURONALES RECURRENTES

Las redes neuronales recurrentes son un tipo de redes en las que, al contrario que las *feed-forward,* la información puede discurrir en otras direcciones que no sea hacia delante. Esto se genera mediante bucles, que permiten conectar las diferentes capas de manera paralela.

Las redes recurrente surgieron a finales del siglo XX, basándose en el trabajo de David Rumelhart y su objetivo es otorgar al usuario la capacidad de conseguir un sistema con temporalidad. Esto quiere decir que los estados anteriores del modelo (paso temporal t-1) pueden afectar a la toma de decisiones en el estado actual (paso temporal t), por lo que se puede decir que las redes neuronales recurrentes tienen memoria.

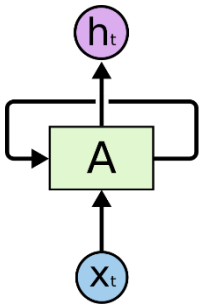

Figura 12. Red neuronal recurrente

2.4.1 Redes neuronales no-lineales autorregresivas con entradas externas (NARX)

Las redes neuronales no-lineales autorregresivas con entradas externas o NARX (por sus siglas en inglés) son un tipo de red neuronal recurrente, que se utiliza en multitud de sistemas dinámicos.

La principal característica de estos modelos es que genera conexiones retroalimentadas entre diferentes capas de la red neuronal. Esto quiere decir que utiliza la capacidad de almacenar los valores obtenidos o reales de tiempos pasados (memoria) para obtener una solución más cercana a la realidad del proceso que se está estudiando.

Como se puede observar en la Figura 13, existen dos tipos de arquitecturas para estos modelos. La primera es la arquitectura *serie-paralelo,* también conocida como de bucle abierto, expresada en la ecuación (19). La segunda se denomina arquitectura *paralela*, también conocida como de bucle cerrado, y se expresa con la ecuación (20).

$$\hat{y}(t+1) = F\begin{pmatrix} y(t), y(t-1), \dots, y(t-n_y) \\ x(t+1), x(t), x(t-1), \dots, x(t-n_x) \end{pmatrix} \quad (19)$$

$$\hat{y}(t+1) = F\begin{pmatrix} \hat{y}(t), \hat{y}(t-1), \dots, \hat{y}(t-n_y) \\ x(t+1), x(t), x(t-1), \dots, x(t-n_x) \end{pmatrix} \quad (20)$$

donde $\hat{y}(t+1)$ es la salida de la red neuronal en un tiempo *t* para un tiempo *t*+1 (la salida en el tiempo presente). $\hat{y}(t), \hat{y}(t-1), \dots, \hat{y}(t-n_y)$ son los valores de las salidas de tiempos anteriores. $y(t), y(t-1), \dots, y(t-n_y)$ son los valores reales del parámetro que se está estudiando, pero al igual que el valor de las salidas, de tiempos pasados. $x(t+1), x(t), x(t-1), \dots, x(t-n_x)$ son las entradas de la red, tanto en el presente como en tiempos anteriores. Por último, n_x y n_y son el número de retrasos utilizados en entradas y salidas, respectivamente (esto hace referencia a cuantos instantes previos quieres tener en cuenta a la hora de realizar la predicción) [5].

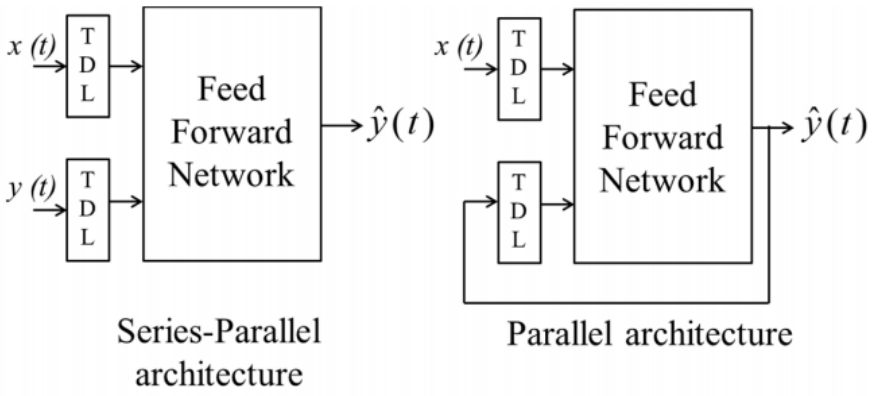

Figura 13. Arquitectura redes NARX

El funcionamiento de la arquitectura *serie-paralelo* consiste en, para obtener un valor de salida de la red en un tiempo cualquiera *t*, se tienen que utilizar como entradas los parámetros de entrada, tanto de ese instante como de los anteriores (hasta llegar al instante marcado por n_x), y los valores reales de los parámetros de salida de momentos pasados (hasta llegar al instante marcado por n_y).

Por otro lado, el funcionamiento de la arquitectura *paralela* es similar al de la arquitectura *serie-paralelo,* pero en vez de usar los valores reales de los parámetros de salida de momentos pasados, se utilizan los valores predichos en la salida de momentos pasados (hasta llegar al instante marcado por n_y).

En este proyecto se van a trabajar con dos modelos distintos de NARX. El primero un modelo NARX basado en Extreme Machine Learning y otro con una arquitectura *serie-paralelo.*

2.4.1.1 NARX basado en ELM

En este primer modelo se quiere conseguir una red neuronal recurrente, pero que a su vez tenga un coste computacional reducido. Por eso, como se ha explicado con anterioridad, el modelo ELM es muy acertado para este propósito.

En este modelo se ha decido usar solamente como entradas los parámetros de entrada, tanto de ese instante como de los anteriores. Excluyendo como variable de entrada cualquier parámetro de la salida de red.

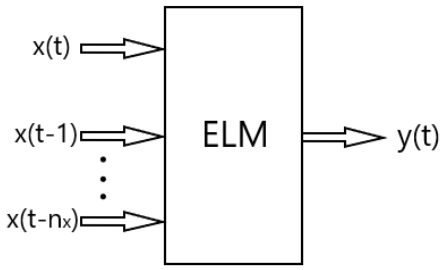

Figura 14. NARX basado en ELM

2.4.1.2 NARX serie-paralelo

Este segundo modelo va a corresponder, como se ha comentado en el apartado anterior a un NARX con arquitectura *serie-paralelo.*

Se ha escogido este modelo porque en este estudio se ha podido acceder a los valores reales de los parámetros de salida, y usar estos valores reales como entradas a la red neuronal otorga más precisión a la predicción.

Otra ventaja es que, de esta manera, la estructura de la red resultante es puramente feed-forward, por lo es posible utilizar algoritmos de entrenamiento de un Perceptrón Multicapa, en este caso el algoritmo utilizado vuelve a ser el de retropropagación [5].

3. METODOLOGÍA

El proyecto que se ha realizado está estructurado en dos grandes partes. En la primera de ellas se ha realizado el estudio estático de los diferentes parámetros del motor y una finalizada esta parte, se ha deseado estudiar el efecto de las dinámicas sobre el comportamiento de la combustión.

El motor sobre el que se ha hecho el estudio es un TJI (Turbulent Jet Ignition). Este tipo de motores reemplazan la típica bujía de encendido por una pequeña cámara de precombustión. Esta cuenta con una bujía y está conectada a la cámara de combustión principal por una serie de orificios que crean unos chorros de productos parcialmente quemados que prenden la cámara principal. Estos chorros de gas caliente (pueden ser entre 4 y 8) generan un efecto de combustión distribuida [7].
Estos motores, al inyectar desde varios puntos simultáneamente da una combustión mucho más uniforme y rápida que en motores convencionales. Esto hace que la eficiencia del motor aumente y además, se consigue una reducción de gases contaminantes como los NOx.

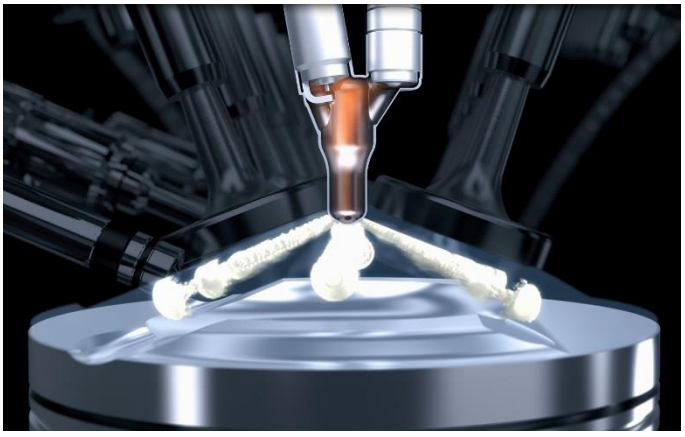

Figura 15. Representación inyección motor TJI

Las características técnicas específicas del motor del ensayo son que se trata de un motor monocilíndrico de cuatro tiempos con una cilindrada de 404cc (centímetros cúbicos) y con dos válvulas de admisión y dos de escape.

3.1 ENSAYO ESTÁTICO

Como punto de partida para este proyecto, se han utilizado tres ensayos realizados previamente sobre el motor que se ha explicado en la introducción de este apartado. En los tres ensayos se ha seguido la misma toma de datos, cambiando únicamente la velocidad de rotación del motor. El primero ha sido con una velocidad de 1350 RPM

(revoluciones por minuto), el segundo con 2000 RPM y por último, el tercero con 3000 RPM.

Los parámetros que se han tomado a través de los distintos sensores instalados en el banco de ensayo donde se llevó a cabo el estudio y con los posteriores cálculos derivados de esa toma de datos son los siguientes:

NOMENCLATURA ENSAYO	DESCRIPCIÓN
T	Tiempo acumulado de cada ciclo
IMEP (Pa)	Presión media efectiva
Max dp	Valor máx. de la der. de presión en cámara
CA 10 (º)	Ángulo cigüeñal al 10% liberación de calor
Ca 50 (º)	Ángulo cigüeñal al 50% liberación de calor
CA 90 (º)	Ángulo cigüeñal al 90% liberación de calor
Pint (Pa)	Presión de admisión
Max p (Pa)	Valor máx. de la presión en cámara
SA (º)	Ángulo adelanta la chispa respecto a PMS.
Fueldur (ms)	Duración de la inyección
mair	Gasto de aire
Lambda (-)	Masa de aire/ Masa de fuel
EGRdes (%)	Valor deseado válvula EGR
EGRpos (%)	Valor real válvula EGR
THRdes (%)	Valor deseado válvula acelerador (Throttle)
THRpos (%)	Valor real válvula acelerador (Throttle)

Tabla 1. Identificación de parámetros

En estos ensayos, el número de muestra que se han tomado es elevado (del orden de 10000) y en algunas de ellas ha habido errores de medidas debidos a problemas en la combustión. Estos problemas son:

- Combustión parcial: esto ocurre cuando no se oxidan todos los componentes del combustible. Suele ser causado por la escasez de comburente, en este caso aire.
- Misfire (fallo inyección): esto ocurre cuando la mezcla de aire y combustible no se enciende dentro del pistón. Esto se debe a la falta de chispa en la bujía.

La detección de estos fallos se ha realizado observando el IMEP a lo largo de todo el ensayo. Como se puede apreciar en la figura 14 hay tres tipos de zonas claramente diferenciadas. La primera, resaltada en color rojo, es el misfire. Esto es así porque al no haber combustión, el IMEP se mantiene a cero. La segunda zona, resaltada en amarillo, es la correspondiente a la combustión parcial, ya que se observan una gran fluctuación de los valores del IMEP. Por último, la zona marcada en verde es en la que se observa una combustión completa, ya que mantiene una valor estable del IMEP a lo largo de un intervalo de muestras.

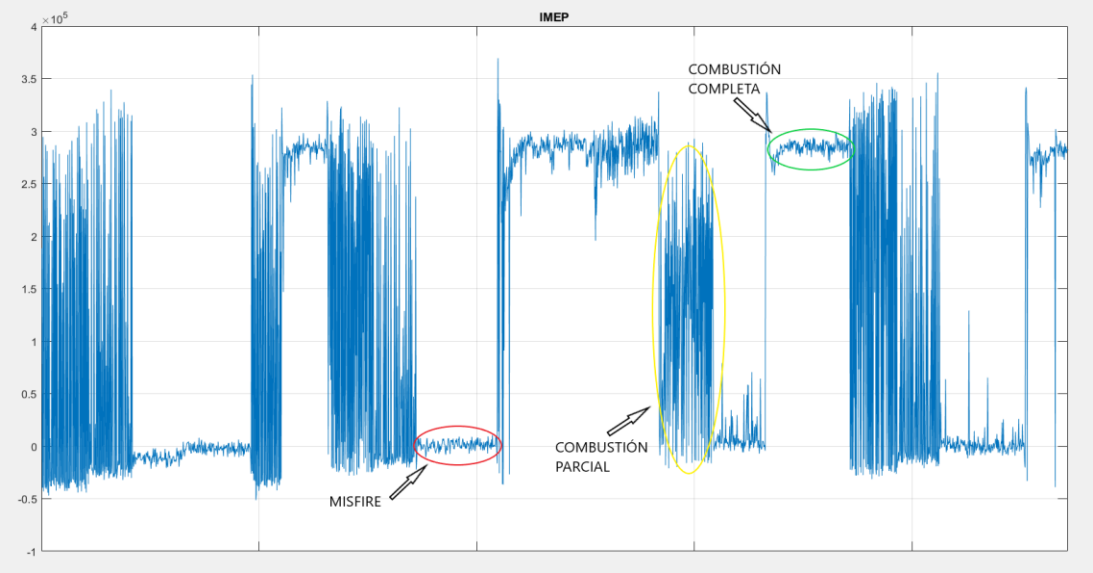

Figura 16. Observación IMEP

Este último tipo de zona son las que interesan a la hora de estudiar el comportamiento del motor, ya que es la única que se puede observar el funcionamiento de este con un comportamiento aceptable.

El procedimiento para la extracción de todos los parámetros de los intervalos en los que hay una combustión completa se puede encontrar en el Anexo 1, como código de Matlab. En primer lugar, para hacer más manejable todo el set de muestras de cada ensayo, se ha decidido subdividir en tramos. Esa subdivisión se realizado de manera arbitraria en los cambios del tiempo de inyección (fueldur). Una vez concretados estos tramos más pequeños, se han representado mediante un set de gráficas conjuntas los tres parámetros medidos directamente del motor, como son el fueldur, el SA y el THRdes, añadiendo el IMEP.

Como se observa en la figura 15, hay gran cantidad de escalones correspondientes a un porcentaje determinado de apertura de la válvula del acelerador (THRdes). Esto valores llevan asociados un tipo de zona de IMEP de los anteriormente explicados. Por lo que observando estas gráficas resulta sencillo conocer todos los intervalos en los que hay combustión completa. Esto se recoge en la primera parte del Anexo 1.

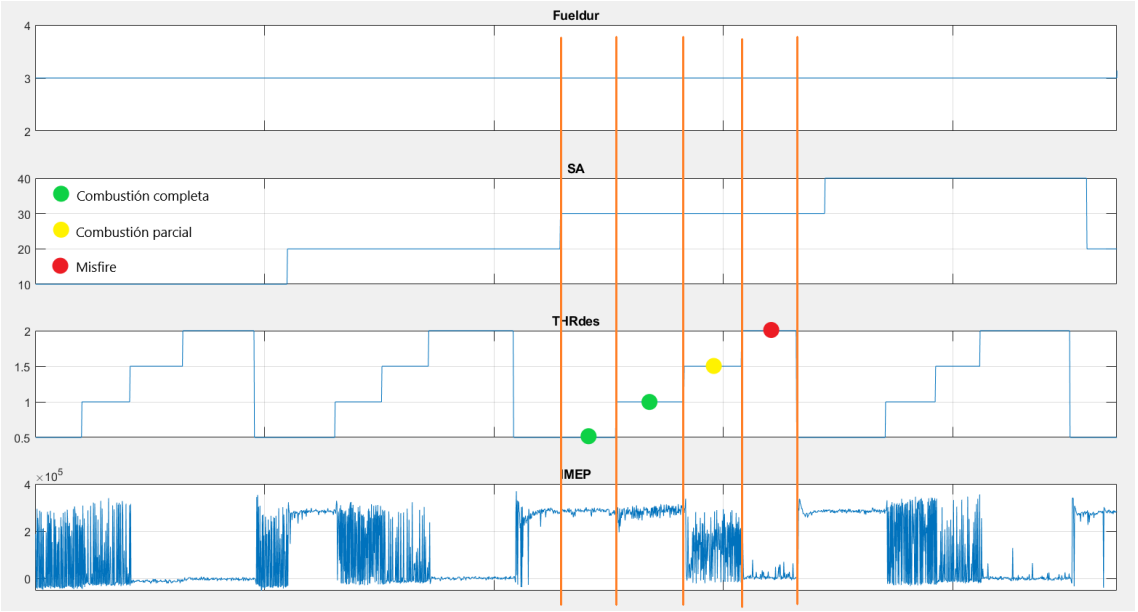

Figura 17. Ejemplo de extracción de datos

Una vez anotados todos los intervalos (punto de inicio y de finalización de cada escalón donde hay combustión completa) de cada una de las subdivisiones de cada ensayo, es el momento de conseguir los valores del resto de parámetros.

Este procedimiento se ha realizado mediante la segunda sección del Anexo 1. En ella se ha generado un bucle *'for'* con el que se ha ido llamando a todos las variables de los intervalos validos que conforman un ensayo y se han guardado en una matriz. Después se ha realizado una media de todos los parámetros que conforman un escalón, para así generar una matriz con dimensiones (M_vxN_m), siendo M_v el número de escalones válidos de un ensayo y N_m el número de variables, quedando introducidas todas las medias representativas de las diferentes variables de cada escalón.

Por último, tras haber finalizado estos pasos para cada uno de los ensayos, se ha procedido a juntar los datos de los tres, para así poder disponer de la información en un solo archivo y permitir trabajar con mayor comodidad. Esto se ve reflejado en la sección 3 del Anexo 1.

Una vez se tiene todos los datos necesarios, es el turno de aplicar los diferentes tipos de redes neuronales para predecir el comportamiento de la combustión. Para ello se va a tratar de realizar la predicción de varios parámetros:

Por un lado, la del IMEP (Indicated Mean Effective Pressure). Este parámetro es esencial para estimar el par motor de motores MCIA, ya que se define con la ecuación (21).

$$IMEP = \frac{W}{V_d} = \cdots = \frac{2\pi T_i * n_r}{n_c * V_d} \quad (21)$$

donde W es el trabajo total de un ciclo completo, V_d es el volumen desplazado, T_i el par motor indicado, n_r es el número de rotaciones del cigüeñal en un ciclo y n_c es el

número de cilindros del motor.
Además, también provee de información relevante acerca de la eficiencia mecánica del ciclo termodinámico que describe la conversión de energía de combustión en trabajo mecánico [6].

Por otro lado, se va a predecir el ca10, ca50 y ca90. El primero marca el ángulo del cigüeñal cuando se ha liberado el 10% del calor acumulado, comúnmente se dice que este es el punto de inicio de la combustión. El segundo marca el ángulo del cigüeñal cuando se ha liberado el 50% del calor acumulado, lo que se conoce como el punto medio de la combustión. Por último, el ca90 marca el ángulo del cigüeñal cuando se ha liberado el 90% del calor acumulado, considerado el punto final de la combustión. Estos parámetros son necesarios para controlar el desempeño de la combustión y para observar como otras variables pueden afectar al retraso o adelanto del inicio y final de misma.

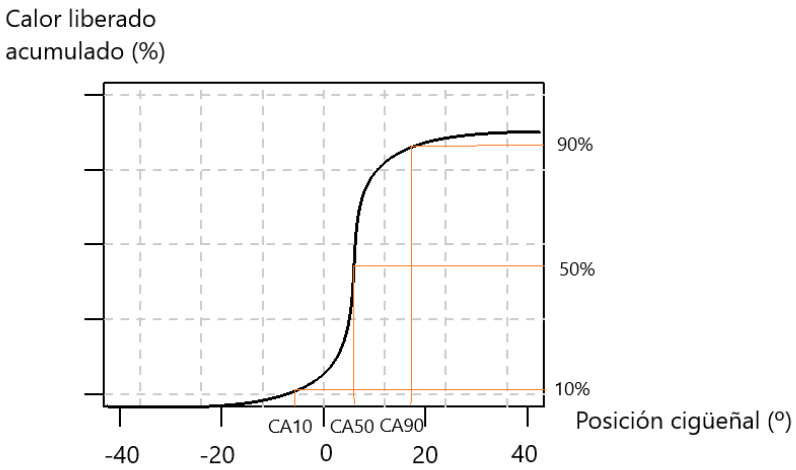

Figura 18. Representación CA10, CA50 y CA90

Las especificaciones de cada una de las redes neuronales se expondrán en el apartado de Resultados, el cual irá a continuación.

3.2 ENSAYO DINÁMICO

Una vez realizado el ensayo estático, se ha decido estudiar cómo se comporta el motor bajo un régimen transitorio. Esto significa que ahora se trata de un sistema dinámico, en el que el tiempo tiene un papel fundamental. Como se puede apreciar en la Figura 17 el modelo estático está basado en una sucesión de intervalos constantes (escalones), en el que el cambio entre ellos ocurre de manera instantánea y como es lógico, esto no se adapta a la realidad.

En la Figura 19 se pueden observar, diferentes parámetros del modelo dinámico. En ellos se aprecia que, a diferencia del modelo estático, los cambios entre los diferentes valores se consiguen de una manera suave y continua.

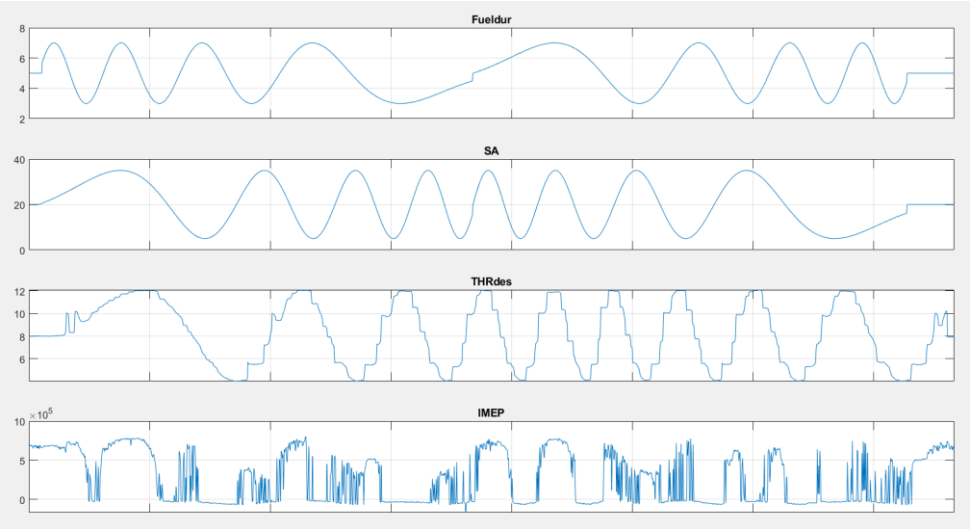

Figura 19. Modelo dinámico

Este avance continuo, en el que no existen saltos entre un estado determinado del motor y otro, se acerca mucho más al comportamiento real de un motor.
Un ejemplo puede ser la válvula del acelerador (Throttle). Esta, en un ensayo estático puede variar su apertura de un 1% a un 5% de manera instantánea, lo cual deja infinidad de puntos intermedios sin control. Mediante un ensayo dinámico se puede estudiar el comportamiento del motor durante este intervalo transitorio, en el que el valor de ese parámetro no es constante. Esto ayuda a conocer el funcionamiento en condiciones más similares a las que se pueden desarrollar en la vida real.

En este modelo se han utilizado dos test de ensayos, el primero tenía una velocidad de 2000 RPM y el 3000 RPM.

4. RESULTADOS

4.1 ENSAYO ESTÁTICO

Para el modelaje de esto ensayos de predicción se han utilizado como datos de entradas el fueldur, el SA, el THRdes y los RPM los cuales son valores extraídos del banco de trabajo directamente. Además, se ha creído conveniente hacer uso de dos parámetros adicionales, como la pint y el mair.

Para medir la actuación de cada uno de los modelos se ha decido utilizar como indicador el error cuadrático medio, que se define como:

$$mse = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (22)$$

siendo n el número de muestras, y_i el valor real de la variable de salida y $\hat{y}_i$ el valor estimado de la variable de salida.

Primero, con la intención de elegir la configuración que mejor se adecuara a nuestra predicción, primero se ha realizado un estudio preliminar sobre el número de neuronas necesarias para la realizar la predicción de manera óptima. En él se ha representado la media de los errores cuadráticos medios según el número de neuronas presentes en la red.

Después se procedido a realizar una gráfica en la que se ha representado, mediante una nube de puntos, la relación entre los valores reales medidos en el ensayo y los valores predichos por nuestra red. En ese mismo gráfico se ha generado una función lineal de pendiente 1, siendo esta la referencia para visualizar la precisión del modelo. Eso significa que si un valor real es dividido por su predicho y ese resultado es 1, el error entre ellos es nulo, por ello cuando más cerca esté la nube de puntos de esa recta, más preciso será el modelo.

4.1.1 Red ELM

La red feed-forward basada en Extreme Machine Learning (ELM) ha sido explicada en el Apartado 2.3 de este proyecto, y ha sido elegida por su gran capacidad de conseguir datos bastante precisos, pero teniendo una baja complejidad computacional, lo que se traduce a poco tiempo de entrenamiento. El código implementado para lo obtención de esta red se puede encontrar en el Anexo 2.

Para este modelo se ha concluido que para minimizar el error del modelo, lo mejor era utilizar un 60% de las muestras para entrenamiento, lo cual ha dejado un 40% para la validación.

En primer lugar, se ha querido realizar la estudio de la configuración del número de neuronas para esta red. Como se puede observar en la Figura 20, la gráfica de los errores respecto al número de neuronas es la siguiente:

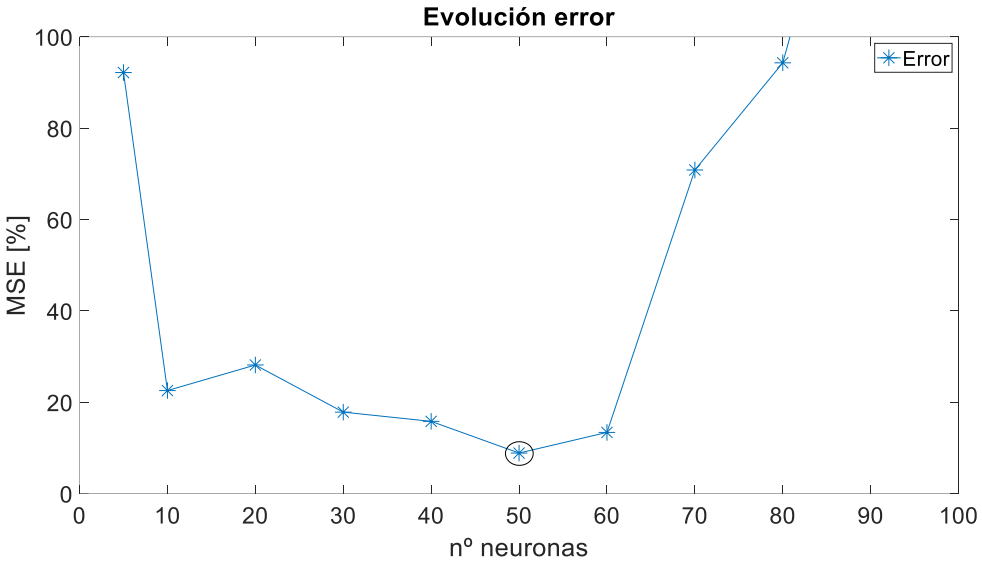

Figura 20. Evolución error con ELM

Por lo que se puede observar, el error mínimo se encuentra en 50 neuronas. Entonces nuestra red va a contar con este número para realizar todo el estudio.

Como primera predicción, se ha procedido a realizar la del IMEP.

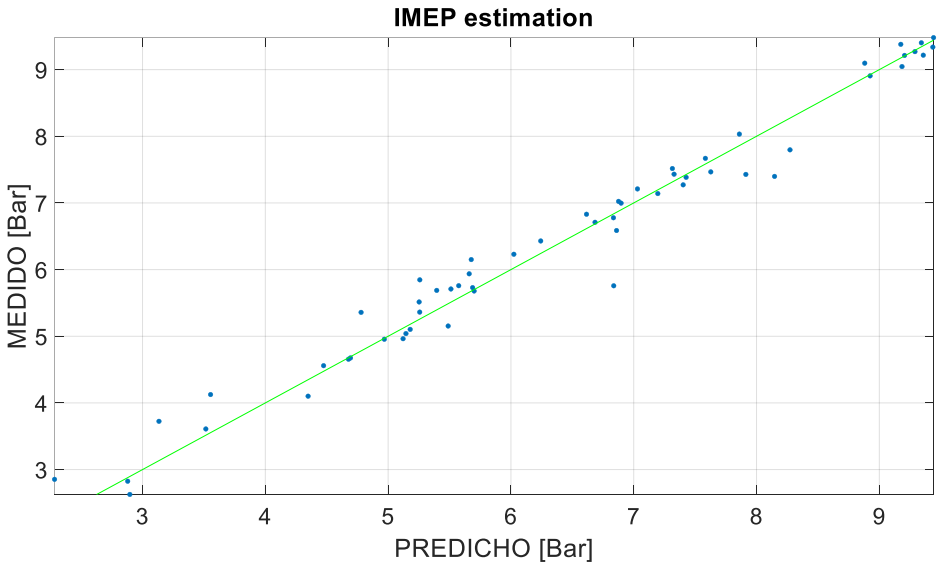

Figura 21. Predicción IMEP con ELM

Tiene un error cuadrático medio (MSE) es de 9,2381%.

Después, se ha procedido a realizar la predicción del CA10.

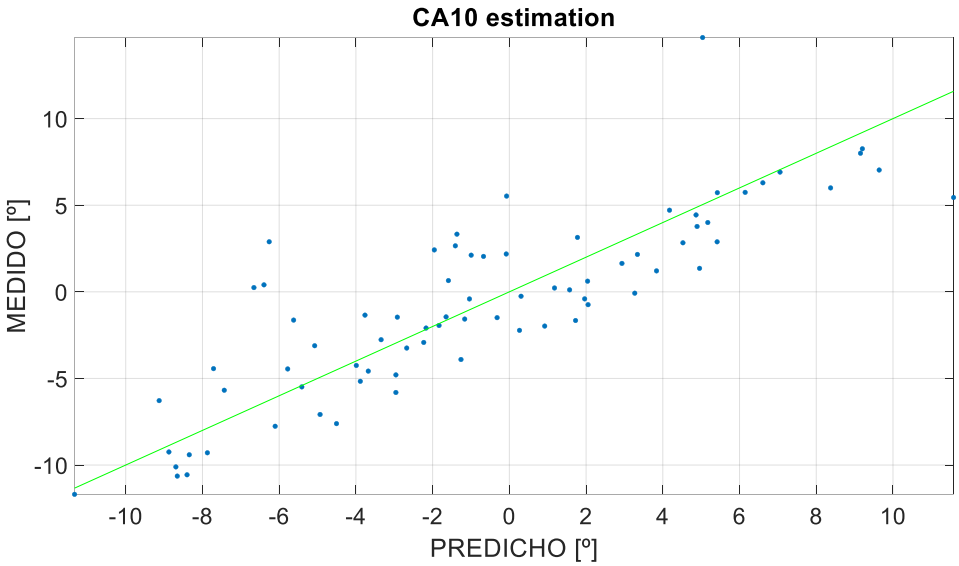

Figura 22.Predicción CA10 con ELM

Tiene un error cuadrático medio (MSE) es de 8,4123%.

En tercer lugar, se ha procedido a realizar la predicción del CA50.

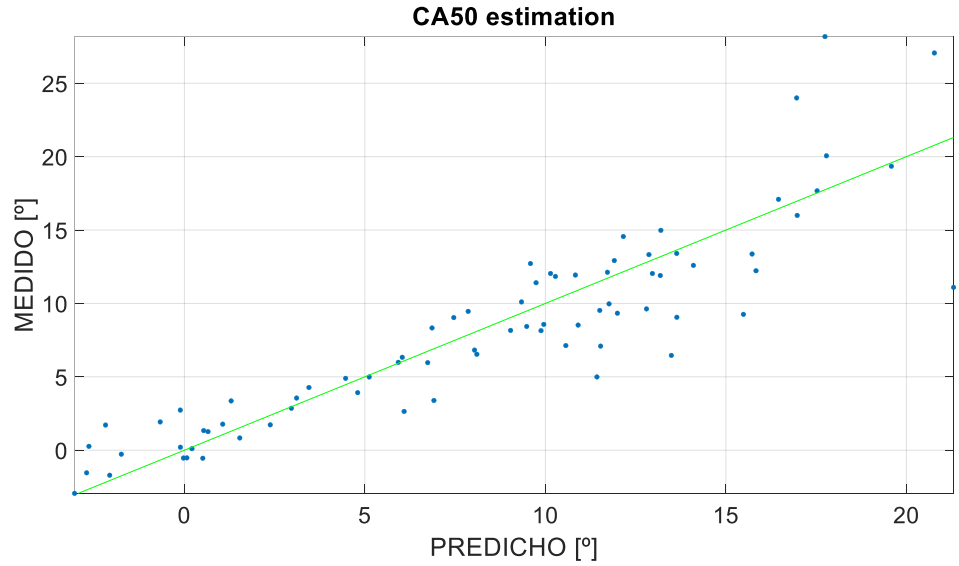

Figura 23.Predicción CA50 con ELM

Tiene un error cuadrático medio (MSE) es de 8,8226%.

Por último, se ha procedido a realizar la predicción del CA90.

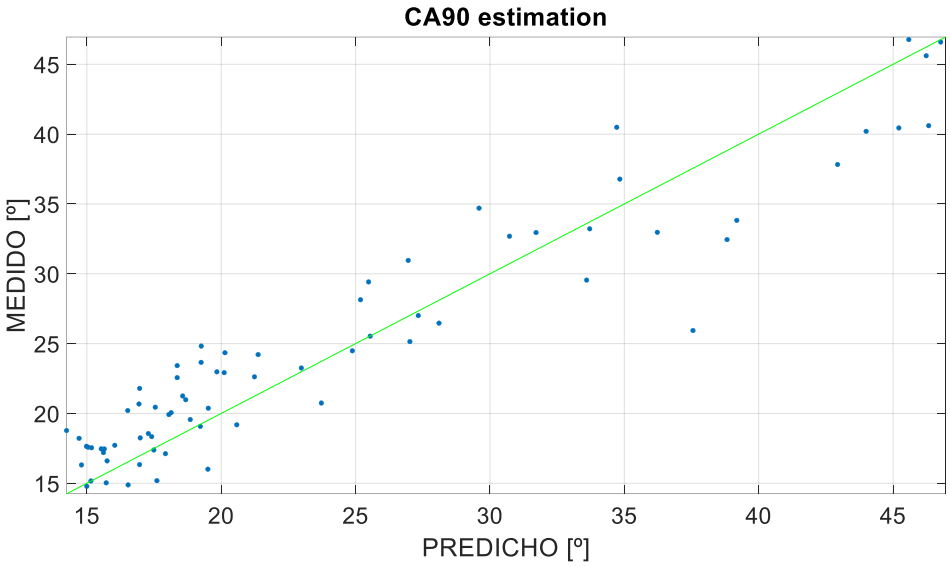

Figura 24. Predicción CA90 con ELM

Tiene un error cuadrático medio (MSE) es de 11,8179%.

El tiempo empleado por esta red ELM para realizar la predicción de todos los parámetros de esta parte del estudio ha sido de 6,63 segundos. Un valor excepcionalmente reducido.

4.1.2 Red MLP con Backpropagation

La red con estructura de perceptrón multicapa y un entrenamiento con el algoritmo de backpropagation (está explicado en el Apartado 2.2.1 de este proyecto) es una de las más utilizadas dentro de este tipo de modelos de aprendizaje automático, ya que te permite conseguir un modelaje con una alta precisión. El mayor inconveniente es el tiempo de entrenamiento, ya que para conseguir un error pequeño es necesario un gran número de iteraciones. El código implementado para lo obtención de esta red se puede encontrar en el Anexo 3.

Al igual que en el modelo anterior, se ha concluido que para minimizar el error del modelo, lo mejor era utilizar un 60% de las muestras para entrenamiento, lo cual ha dejado un 40% para la validación.

En primer lugar, se ha querido realizar la estudio de la configuración del número de neuronas para esta red. Como se puede observar en la Figura 25, la gráfica de los errores respecto al número de neuronas es la siguiente:

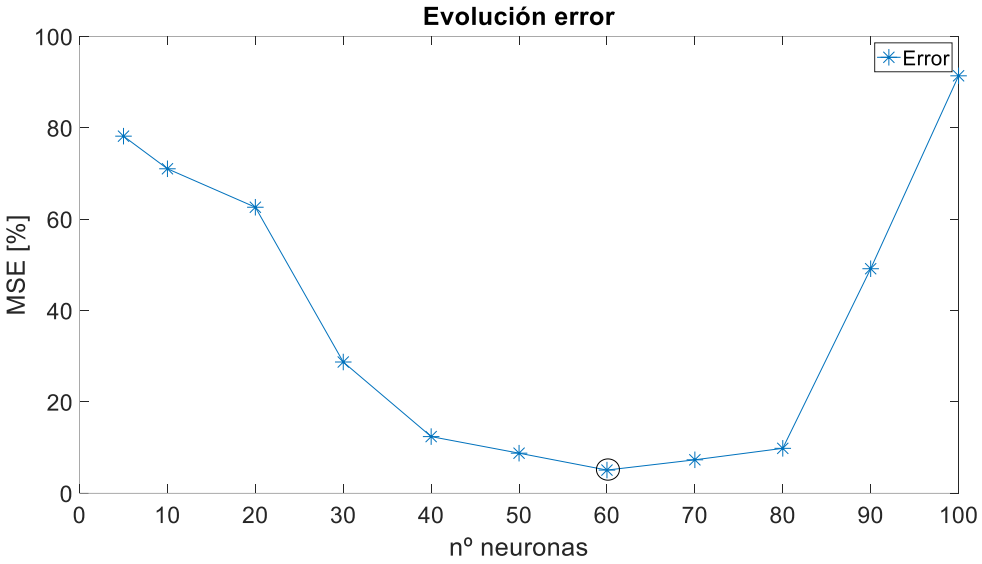

Figura 25. Evolución error con MLP-BP.

Por lo observado en la gráfica anterior, en este caso se ha decidido utilizar una red MLP con 60 neuronas en la capa oculta.

Al tener claro el número de neuronas en la capa oculta, se ha realizado la predicción del primer parámetro, en este caso el IMEP.

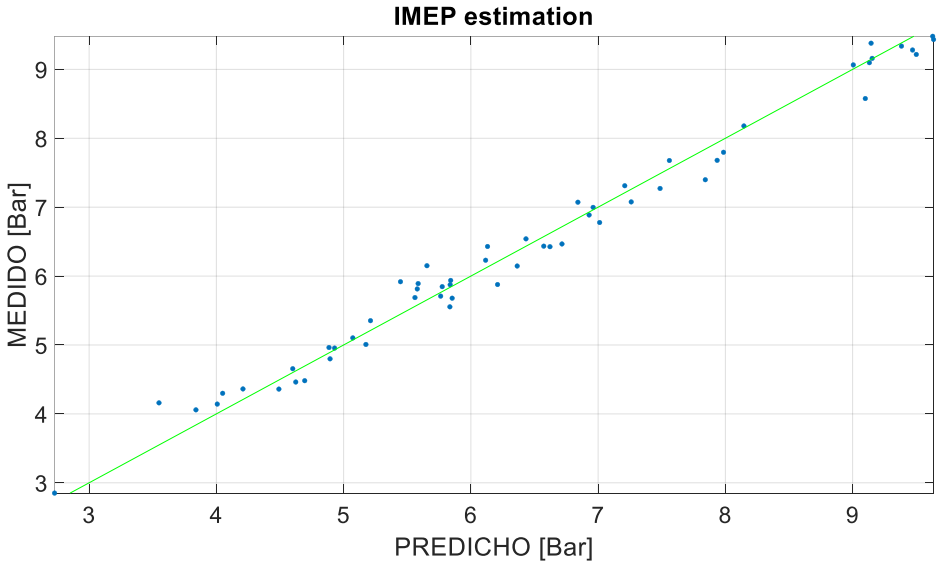

Figura 26. Predicción IMEP con MLP-BP

Tiene un error cuadrático medio (MSE) es de 5,0506%.

Después, se ha procedido a realizar la predicción del CA10.

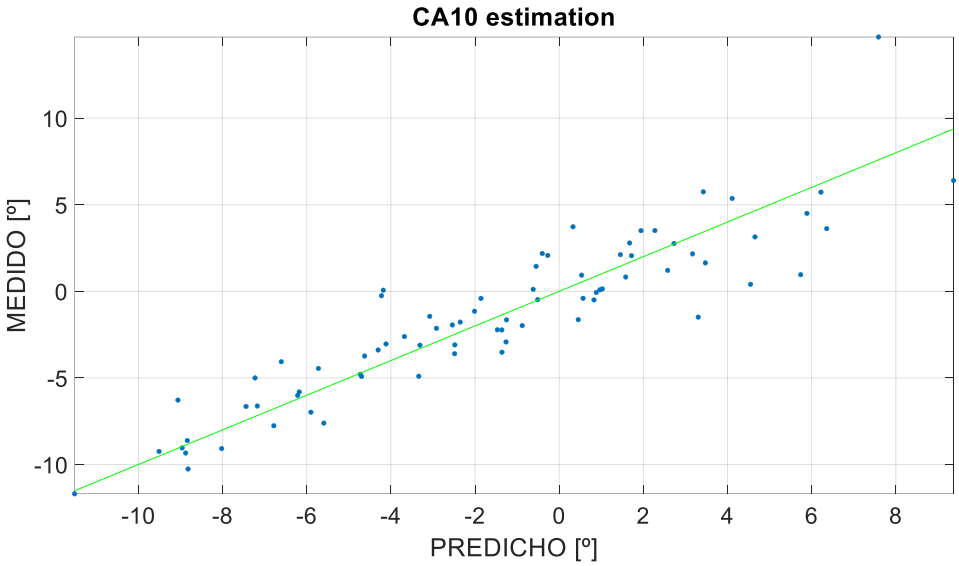

Figura 27. Predicción CA10 con MLP-BP

Tiene un error cuadrático medio (MSE) es de 3,7085%.

Más tarde, se ha procedido a realizar la predicción del CA50.

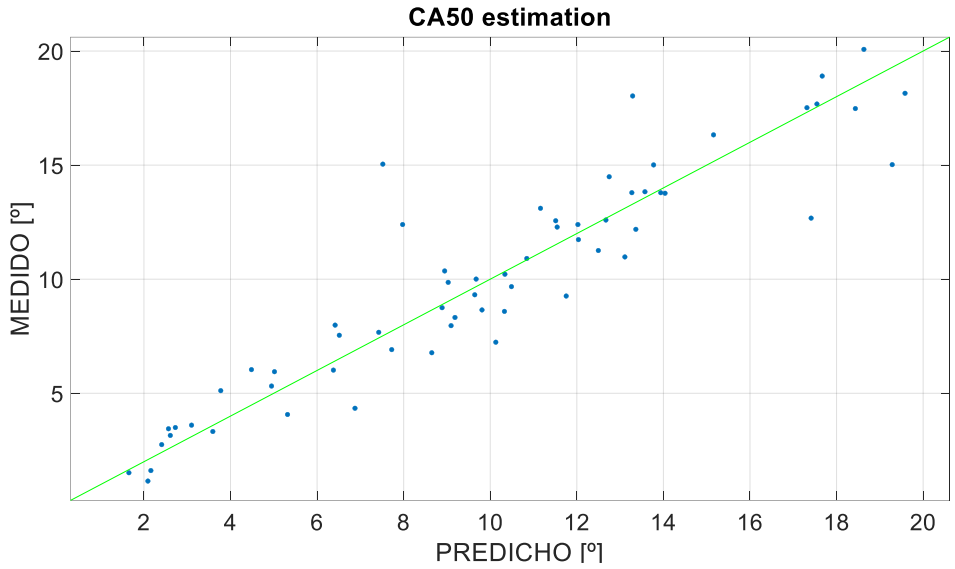

Figura 28. Predicción CA50 con MLP-BP

Tiene un error cuadrático medio (MSE) es de 5,5889%.

Por último, se ha procedido a realizar la predicción del CA90.

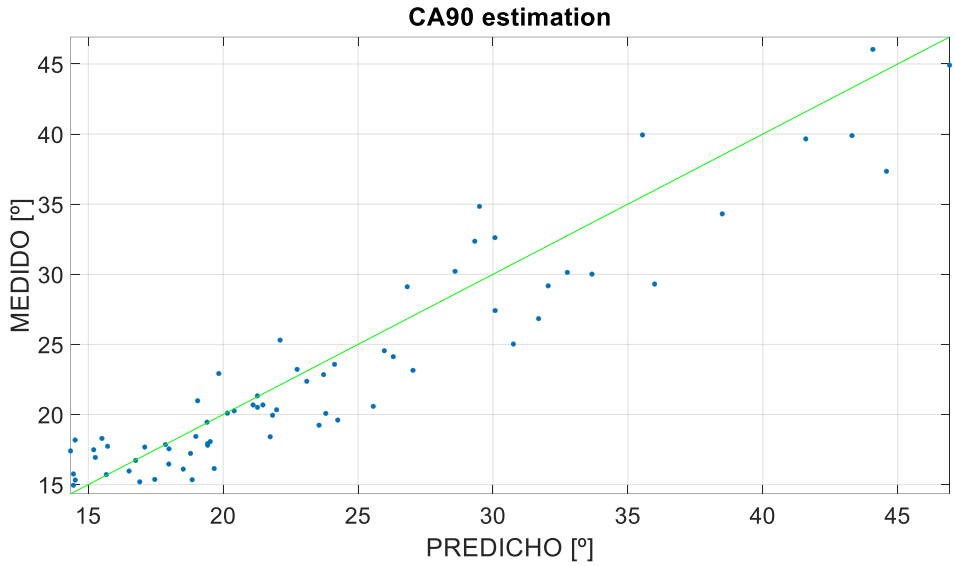

Figura 29. Predicción CA90 con MLP-BP

Tiene un error cuadrático medio (MSE) es de 8,4481%.

El tiempo de entrenamiento que ha necesitado esta red para encontrar una solución satisfactoria ha sido de 118,72 segundos. Sustancialmente mayor que la de la red anteriormente planteada.

4.1.3 Comparación de Modelos

La manera que se ha escogido para comparar ambos modelos ha sido generar los histogramas del error. En ellos se ve reflejado como se agrupan los errores con respecto al error cero. Esto ejemplifica de manera muy visual la distribución de este parámetro, siendo un indicador muy eficaz para evaluar la actuación de estos.

Estos histogramas se han adjuntado de manera continuada según la variable estudiada, para así poder evaluarlas de manera individual pero también poder compararlas de manera conjunta.

Como datos adicionales a esta comparativa, se han calculado el valor medio de las diferencias de cada una de las redes, así como su desviación típica. Estos datos pueden ayudar a esclarecer el comportamiento del modelo, ya que te indican a que distancia dista la media del valor ideal (el cero en este caso) y la dispersión de la totalidad de los datos.

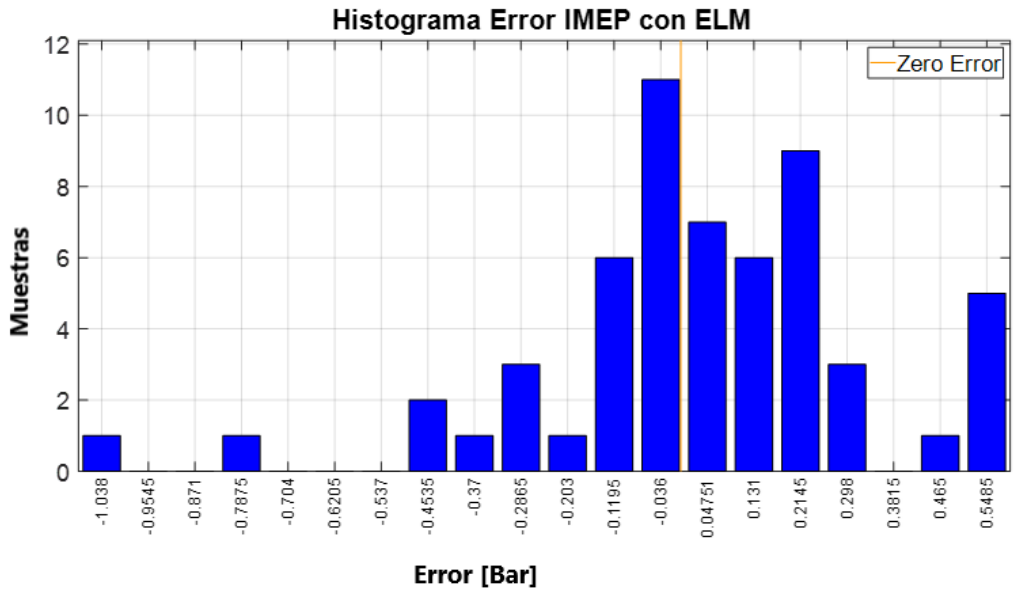

Figura 30. Histograma error IMEP con ELM

La media de la diferencia de valores del IMEP en la red ELM ha sido 0,0319 y la desviación típica de 0,3050.

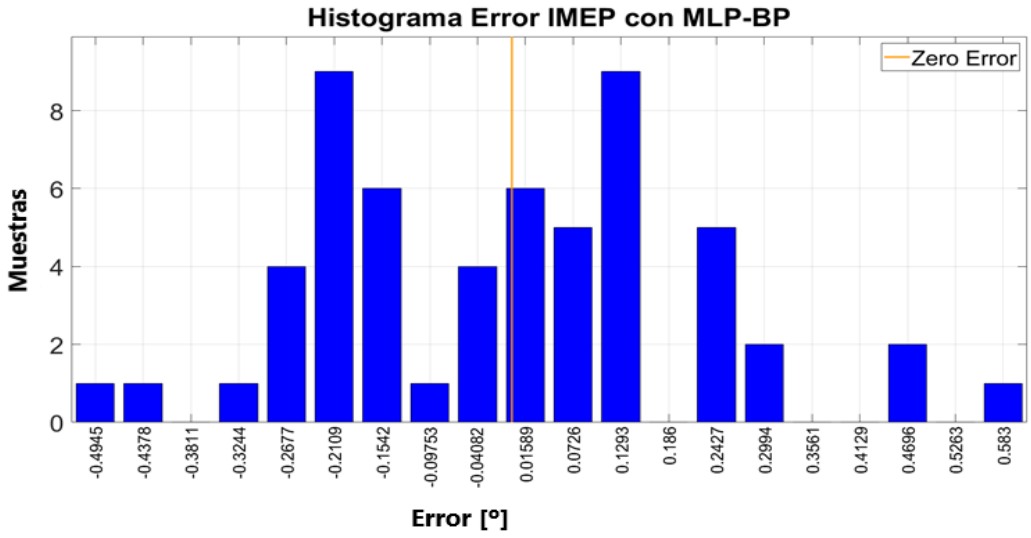

Figura 31. Histograma error IMEP con MLP-BP

La media de la diferencia de valores del IMEP en la red MLP-BP ha sido -0,0069 y la desviación típica de 0,2266.

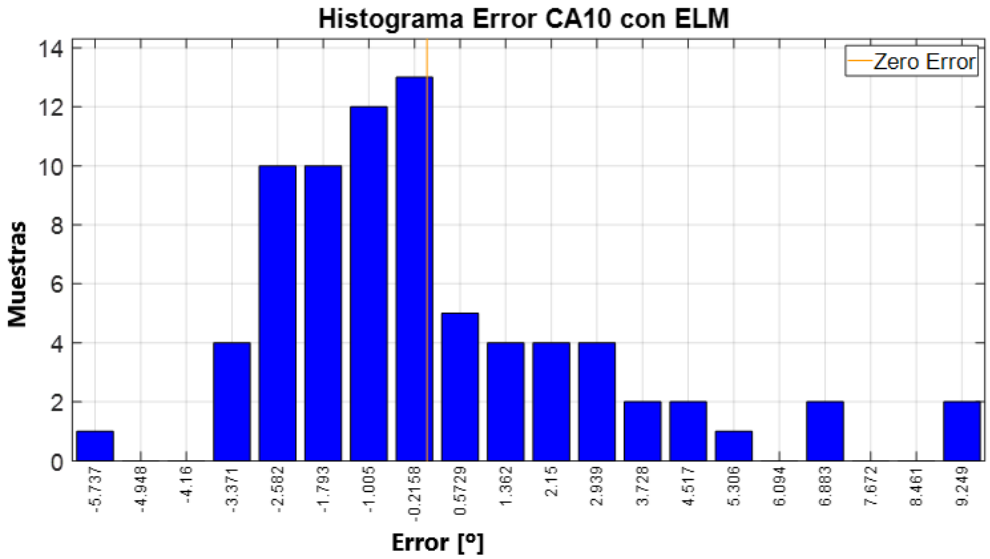

Figura 32. Histograma error CA10 con ELM

La media de la diferencia de valores del CA10 en la red ELM ha sido 0,0587 y la desviación típica de 2,9191.

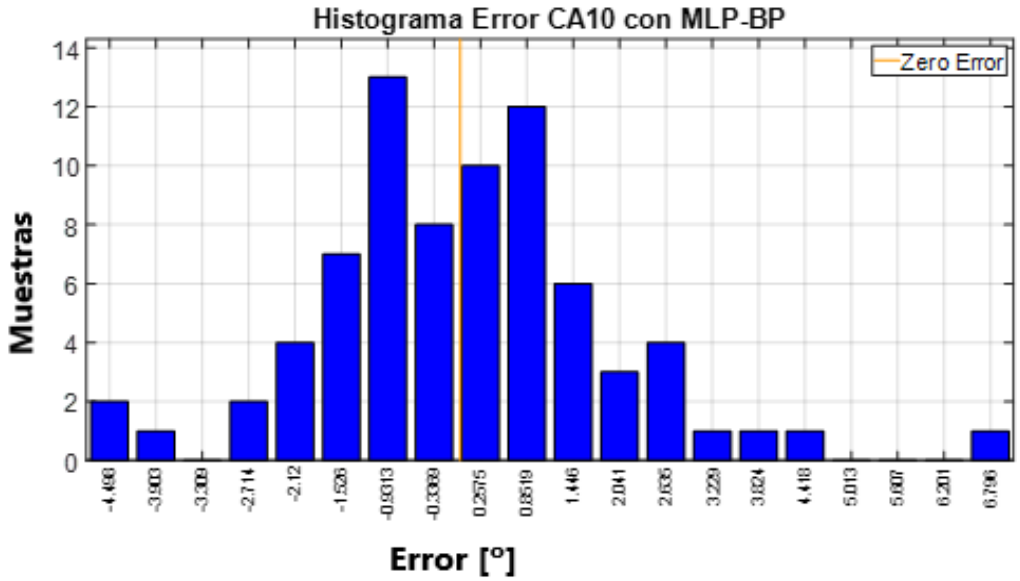

Figura 33. Histograma error CA10 con MLP-BP

La media de la diferencia de valores del CA10 en la red MLP-BP ha sido 0,0510 y la desviación típica de 1,9379.

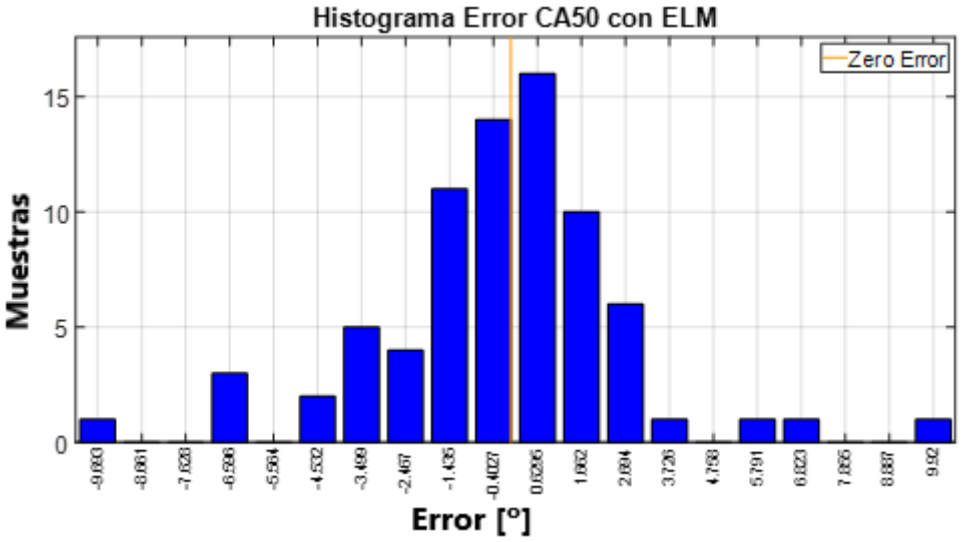

Figura 34. Histograma error CA50 con ELM

La media de la diferencia de valores del CA50 en la red ELM ha sido -0,2125 y la desviación típica de 2,9824.

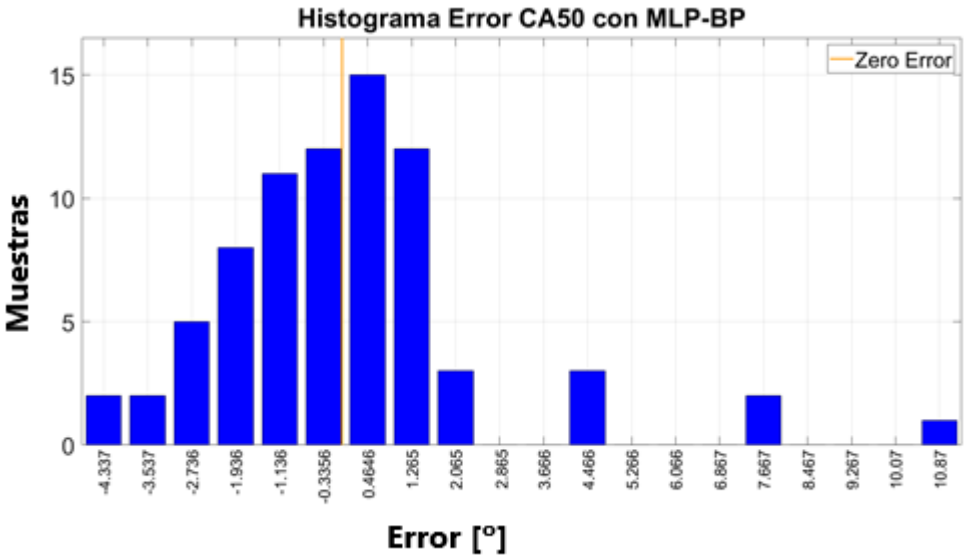

Figura 35. Histograma error CA50 con MLP-BP

La media de la diferencia de valores del CA50 en la red MLP-BP ha sido 0,0945 y la desviación típica de 2,5312.

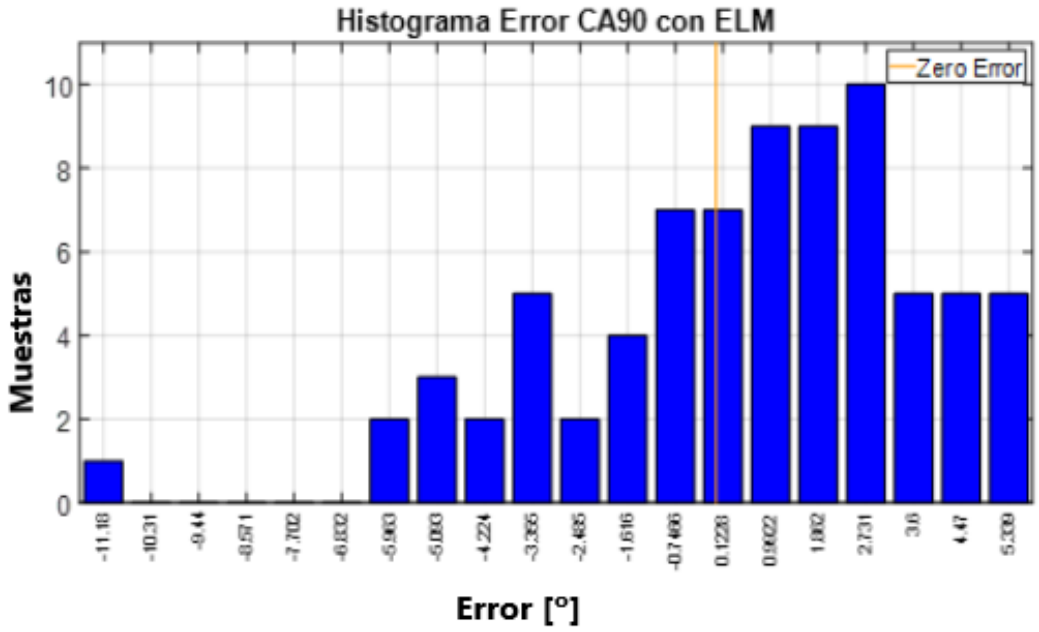

Figura 36. Histograma error CA90 con ELM

La media de la diferencia de valores del CA90 en la red ELM ha sido 0,5665 y la desviación típica de 3,2615.

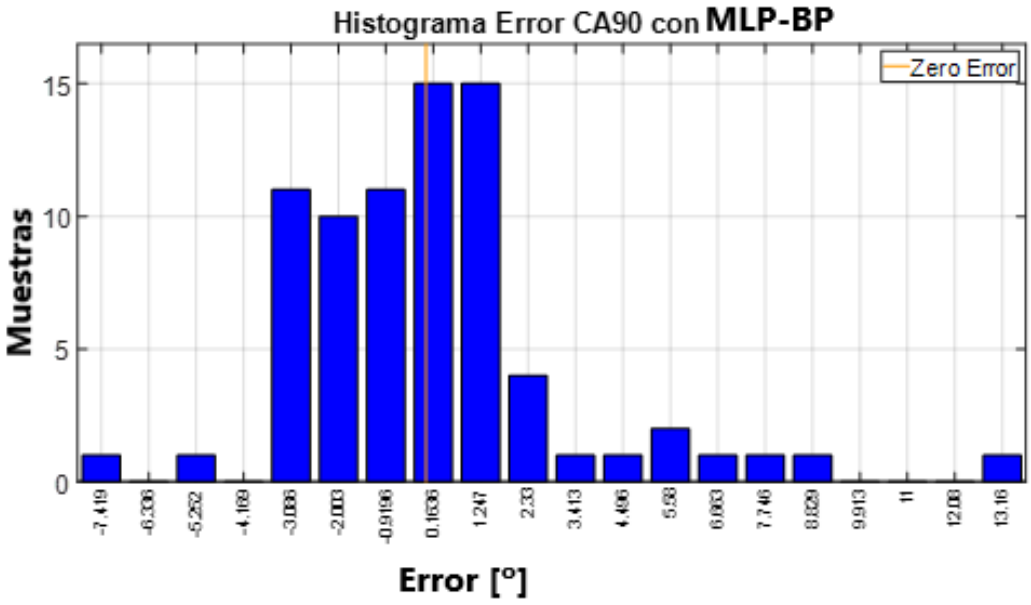

Figura 37. Histograma error CA90 con MLP-BP

La media de la diferencia de valores del CA90 en la red MLP-BP ha sido 0,1361 y la desviación típica de 3,1509.

A modo de resumen se van a realizar tablas para poder consultar de manera más sencilla y poder observar la actuación cada red en función de diferentes variables de medida.

IMEP

	Mse [%]	Media	Desviación típica
ELM	9,2381	0,0319	0,3050
MLP	5,0506	-0,0069	0,2266

Tabla 2. Resumen IMEP Estático

CA10

	Mse [%]	Media	Desviación típica
ELM	8,4123	0,0587	2,9191
MLP	3,7085	0,0510	1,9379

Tabla 3. Resumen CA10 Estático

CA50

	Mse [%]	Media	Desviación típica
ELM	8,8226	-0,2125	2,9824
MLP	5,5889	0,0945	2,5312

Tabla 4. Resumen CA50 Estático

CA90

	Mse [%]	Media	Desviación típica
ELM	11,8179	0,5665	3,2615
MLP	8,4481	0,1361	3,1509

Tabla 5. Resumen CA90 Estático

Tiempo de entrenamiento

	Tiempo [s]
ELM	6,63
MLP	118,72

Tabla 6. Tiempo entrenamiento Estático

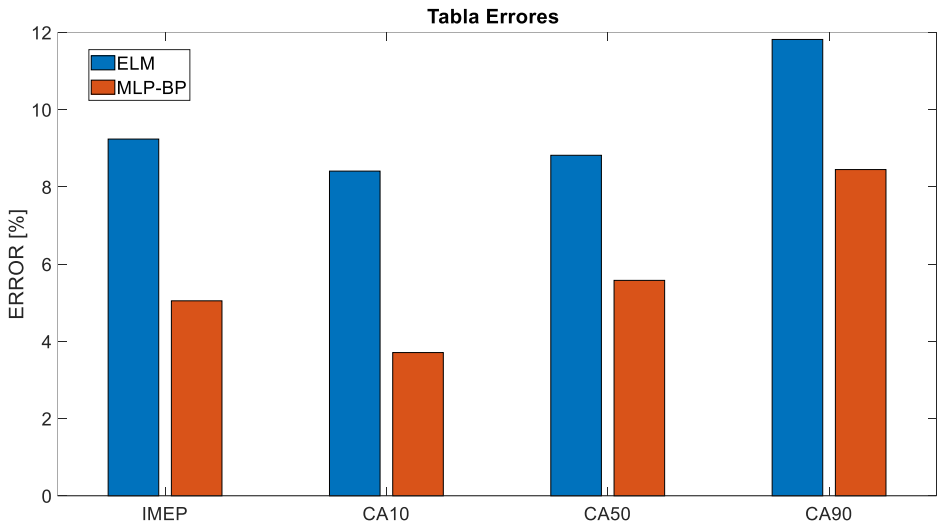

Figura 38. Diagrama errores Estático

4.2 ENSAYO DINÁMICO

Para el modelaje dinámico se decidido volver a utilizar los mismos parámetros de entrada que en el ensayo estático, por lo que una vez más estos van a ser el tiempo de inyección (fueldur), el SA, el THRpos, los RPM, la pint y la mair.

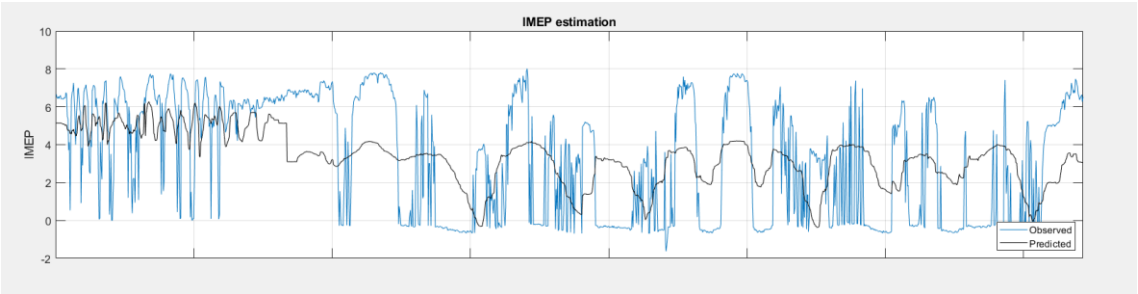

Figura 39. Modelo dinámico con red feed-forward

Como se puede observar en la Figura 39 si se intenta realizar la predicción de un modelo dinámico mediante una red neuronal feed-forward esta no consigue realizarla de manera correcta, ya que no consigue captar la esencia de este. Por ello, para conseguir predecir de manera acertada este modelo es necesario recurrir a las redes neuronales recurrentes, explicadas en el Apartado 2.4.

Para la elección del número de neuronas en la capa se ha seguido el mismo procedimiento que en el ensayo estático, ya que se ha generado una gráfica donde se recogen los errores medios según el número de estas.

Para realizar la predicción de las dinámicas se ha creído conveniente realizar una gráfica en la que se representen los valores reales y predichos a través de los ciclos. Mediante este método se va a poder apreciar cómo se ajusta el modelo a la realidad y así poder decidir si es las redes recurrentes son un método fiable para estudiar los estados transitorios del motor.

Al igual que en el ensayo anterior, se ha decido utilizar un 60% de las muestras para el entrenamiento de las redes neuronales NARX, por lo que para la validación se han destinado un 40%.

4.2.1 Red NARX-ELM

Esta red neuronal recurrente NARX está basada en el Extreme machine Learning (ELM), con el objetivo de que conseguir hacer una predicción aceptable, pero con un coste computacional bajo, siendo esta la principal ventaja de los modelos ELM. El código implementado para lo obtención de esta red se puede encontrar en el Anexo 4.

Como bien se ha comentado anteriormente en este proyecto, esta red solo va a utilizar como entradas, los inputs del instante de estudio como de momentos anteriores. Después de probar varias configuraciones, se ha decido utilizar decidido utilizar solo el instante anterior como entrada, ya que si se añadían más instantes el error aumentaba ligeramente (4% aproximadamente).

Un esquema ejemplificativo de la configuración escogida para esta red es el que se refleja en la figura 40.

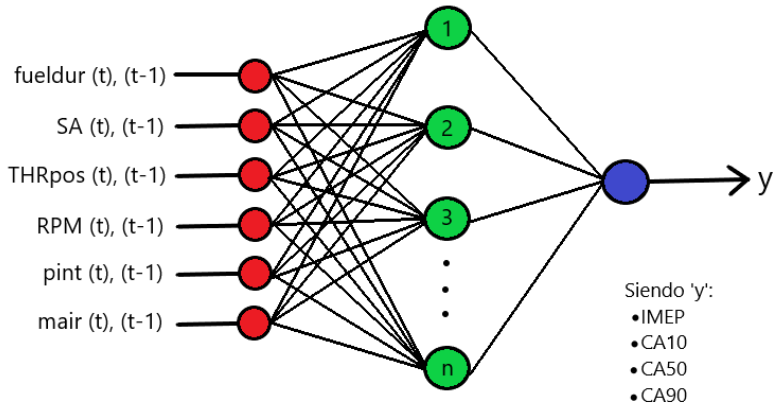

Figura 40. Esquema red NARX-ELM

En primer lugar, se ha querido realizar la estudio de la configuración del número de neuronas para esta red. Como se puede observar en la Figura 41, la gráfica de los errores respecto al número de neuronas es la siguiente:

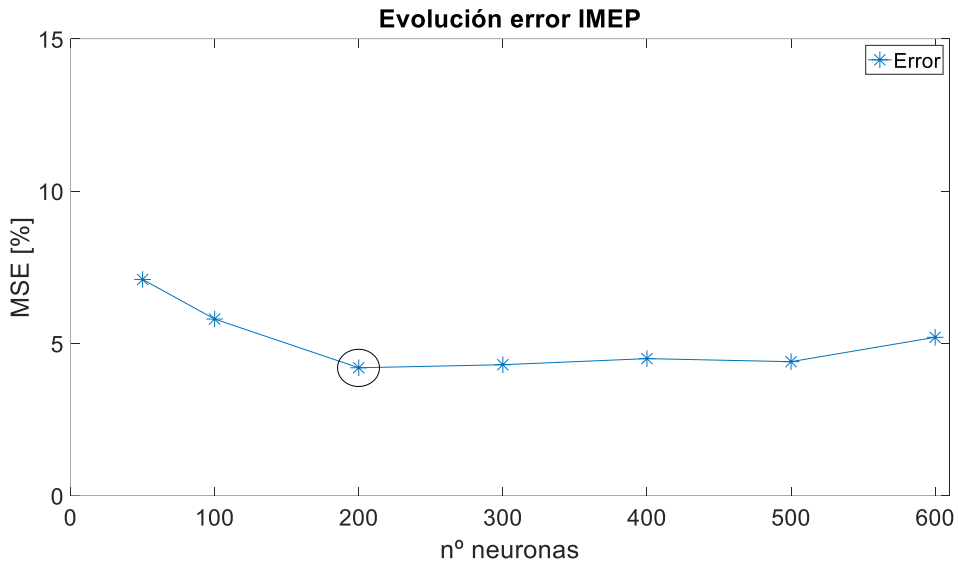

Figura 41. Evolución error con NARX-ELM

Por lo que se puede ver en la gráfica anterior, se ha decidido utilizar una red NARX-ELM con 200 neuronas en la capa oculta.

Primero, se ha realizado la predicción del IMEP.

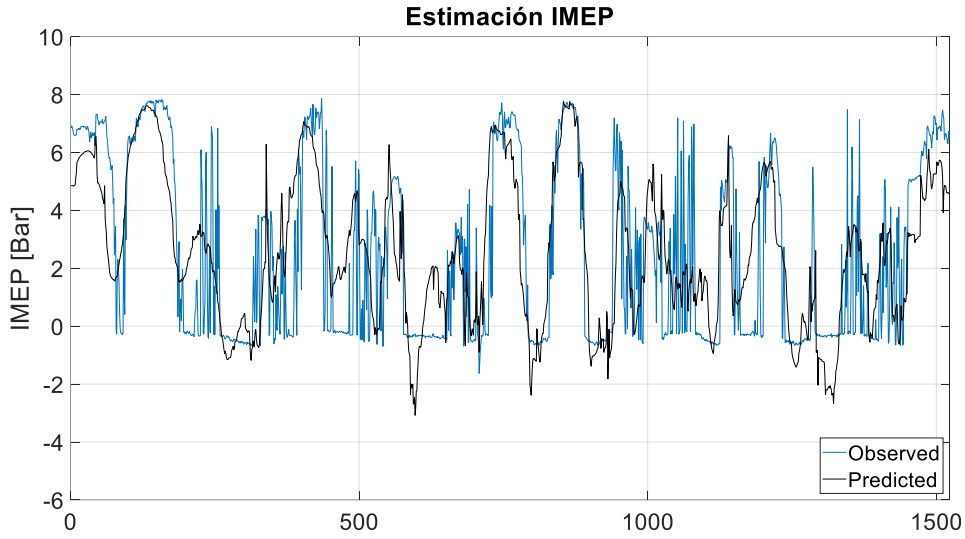

Figura 42. Predicción IMEP con NARX-ELM

Tiene un error cuadrático medio (MSE) es de 4,2299%.

Una vez realizada esta predicción, se ha procedido a realizar el mismo proceso pero esta vez para el parámetro CA10.

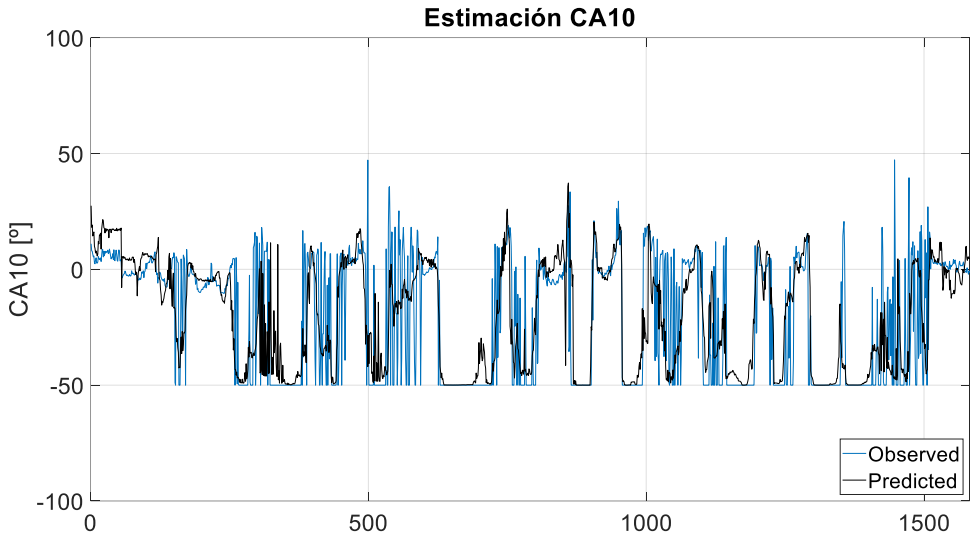

Figura 43. Predicción CA10 con NARX-ELM

Tiene un error cuadrático medio (MSE) es de 8,2956%.

Acto seguido, se ha realizado la predicción de CA50.

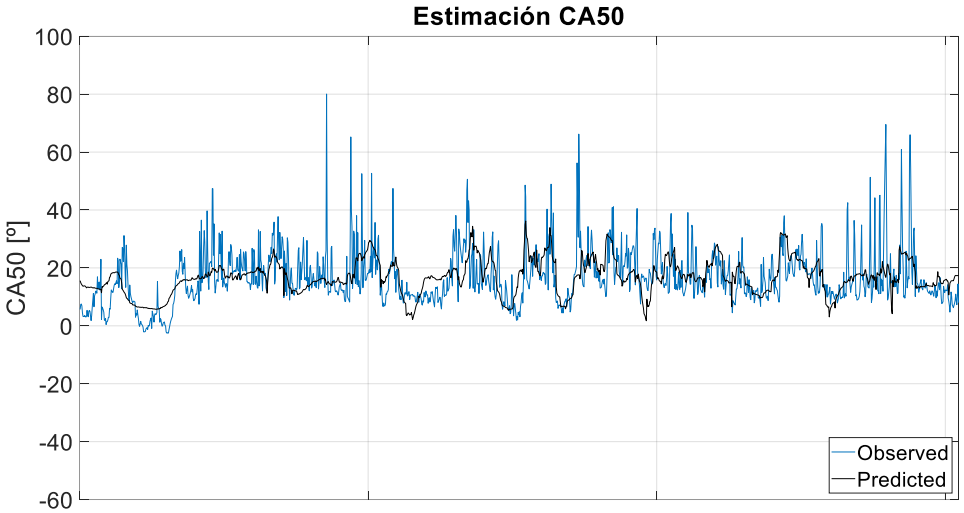

Figura 44. Predicción CA50 con NARX-ELM

Tiene un error cuadrático medio (MSE) es de 9,3295%.

Por último, se vuelto a repetir el mismo procedimiento para la consecución del parámetro CA90.

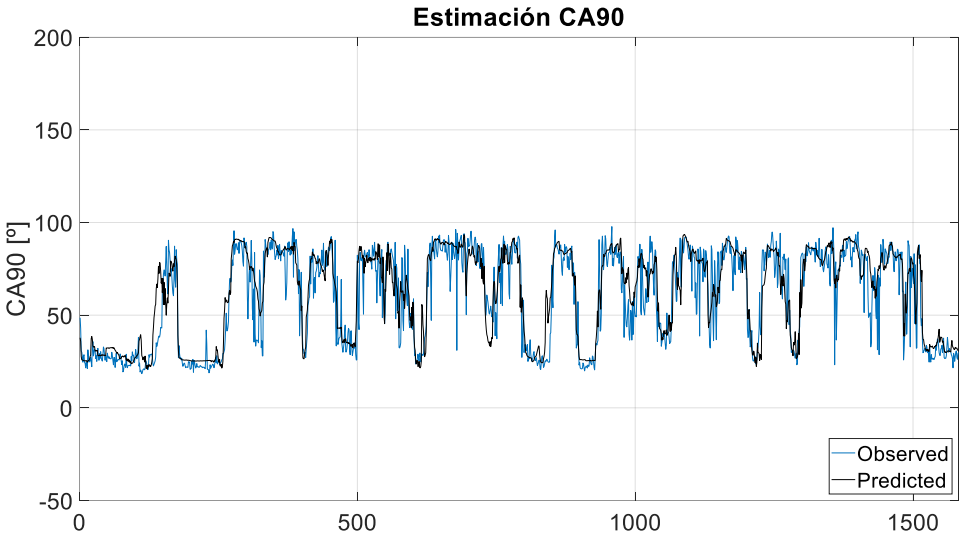

Figura 45. Predicción CA90 con NARX-ELM

Tiene un error cuadrático medio (MSE) es de 7,1280%.

El tiempo de entrenamiento empleado por esta red para realizar la predicción ha sido de 11,27 segundos.

4.2.2 Red NARX con Backpropagation

Esta red neuronal recurrente NARX tiene una arquitectura *serie-paralelo,* la cual utiliza como entradas, los inputs del instante de estudio y de momentos anteriores, así como los valores reales anteriores de la variable de salida. Solo se ha añadido un instante anterior para ambos parámetros, ya que después de un estudio preliminar esta configuración era la que mejor se adecuaba a este estudio.

Como se ha explicado con anterioridad, el algoritmo de entrenamiento utilizado ha sido el de Backpropagation, el cual permite resultados precisos. El código implementado para lo obtención de esta red se puede encontrar en el Anexo 5.

El número de muestras utilizadas para el entrenamiento han sido el 60%, ya que era lo ideal para mejor el rendimiento de la red.

Por último, se ha decidido que para utilizar 3 capas ocultas. Para ejemplificarlo se va a utilizar una tabla en la que se ha partido de un error del 10% en entrenamiento en todos los casos (probado sobre los datos del IMEP). En ella se va a rellenar el número de iteraciones necesarias, el error obtenido en validación y el tiempo requerido.

	iteraciones	% error validación	Tiempo (s)
1 capa	---	---	---
2 capas	53500	2,46	209
3 capas	31200	1,95	156
4 capas	33400	1,93	215

Tabla 7. Estudio del número de capas NARX-BP

Observando la tabla se puede apreciar que en 3 capas se obtiene el error pequeño pero el menor tiempo. Por ello, a partir de las 3 capas el tiempo de computación no compensaba el error obtenido.

Como el número de capas ocultas es alto y las configuraciones con diferente número de neuronas en cada capa es muy elevada se ha decido utilizar 50 neuronas en cada capa.

Un esquema ejemplificativo de la configuración escogida para esta red es el que se refleja en la figura 46.

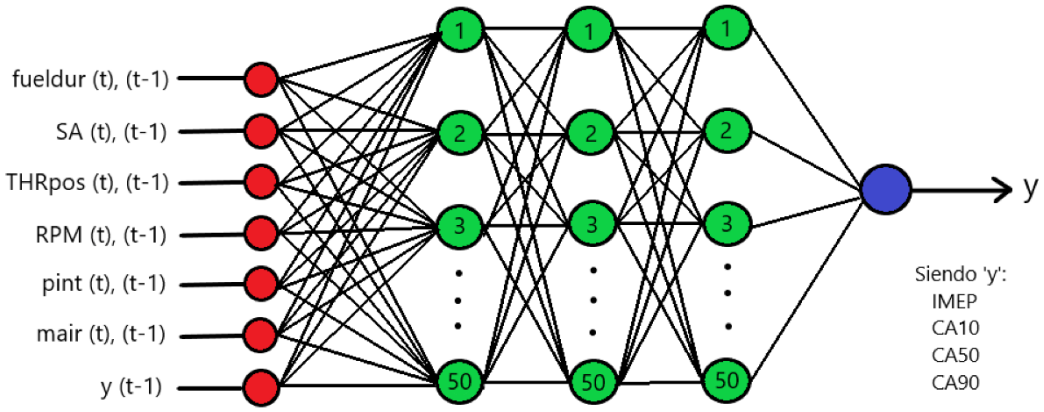

Figura 46. Esquema red NARX-BP

En primer lugar, se ha querido realizar la predicción del IMEP de este modelo dinámico.

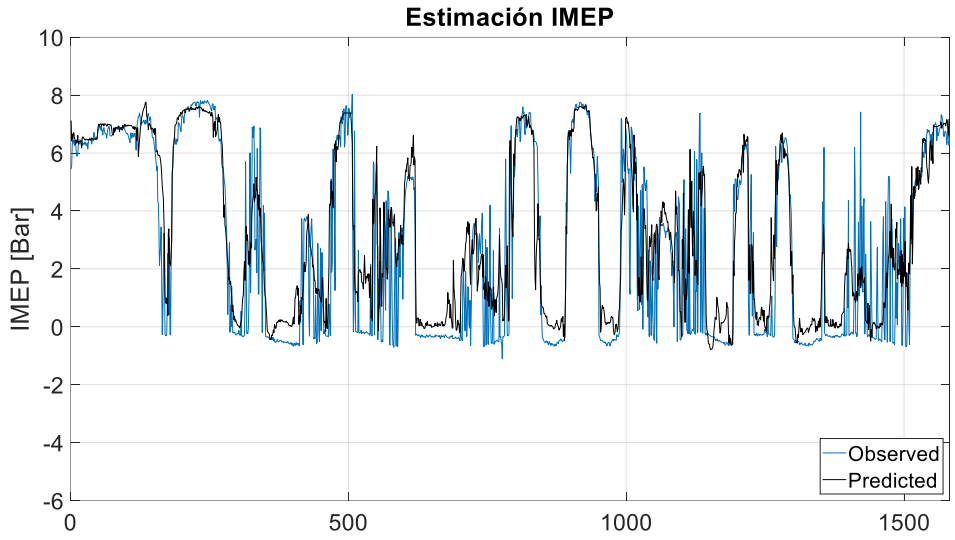

Figura 47. Predicción IMEP con NARX-BP

Tiene un error cuadrático medio (MSE) es de 1,9491%.

Una vez realizada esta predicción, se ha procedido a realizar el mismo proceso pero esta vez para el parámetro CA10.

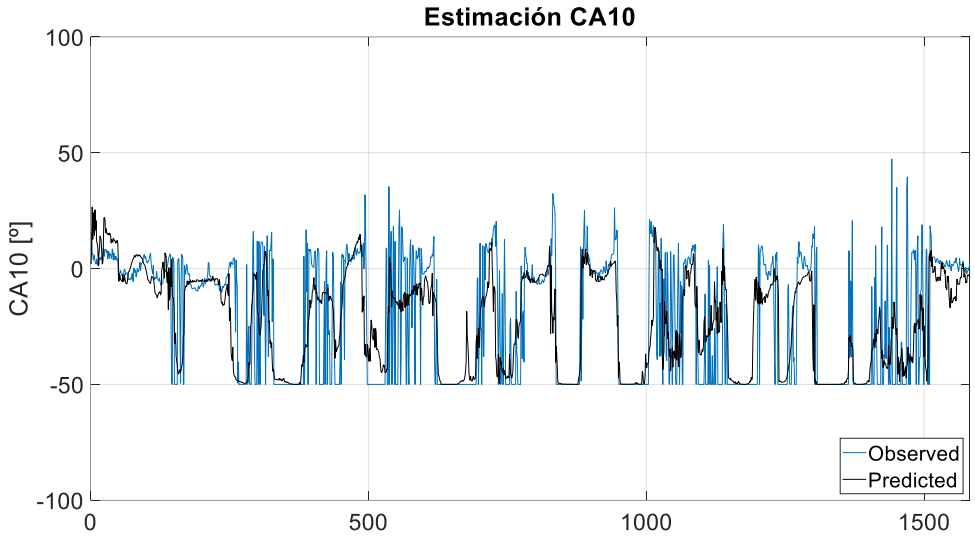

Figura 48. Predicción CA10 con NARX-BP

Tiene un error cuadrático medio (MSE) es de 6,6471%.

Acto seguido, se ha realizado la predicción de CA50, mediante el mismo proceso que en las variables anteriores.

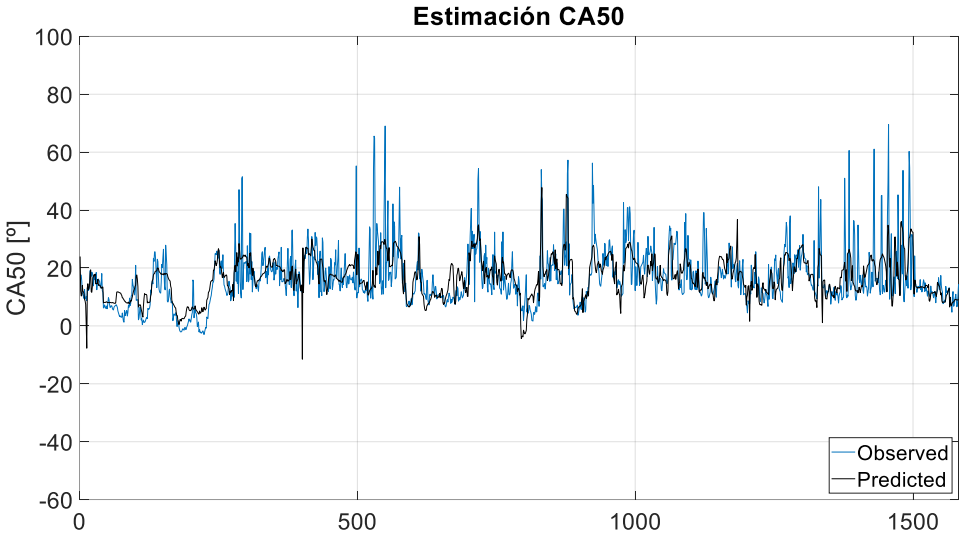

Figura 49. Predicción CA50 con NARX-BP

Tiene un error cuadrático medio (MSE) es de 7,7104%.

Por último, se vuelto a repetir el mismo procedimiento para la consecución del parámetro CA90.

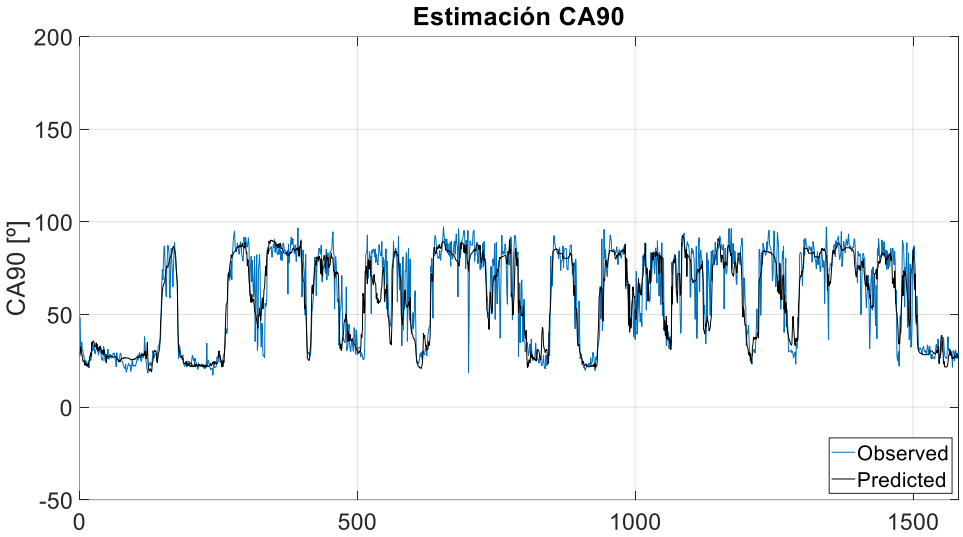

Figura 50. Predicción CA90 con NARX-BP

Tiene un error cuadrático medio (MSE) es de 5,3745%.

El tiempo de entramiento de esta red a la hora de realizar la totalidad de las predicciones necesarias para esta parte del estudio ha sido de 542,05 segundos.

4.2.3 Comparación de Modelos

Para realizar la comparativa de la actuación de las dos redes recurrentes se ha procedido de la misma manera que en el ensayo estático, es decir, representar las diferencias entre los valores reales y los predichos de ambos modelos para así poder ser comparados. También, al igual que en el proceso anterior, se han calculado la media y la desviación típica.

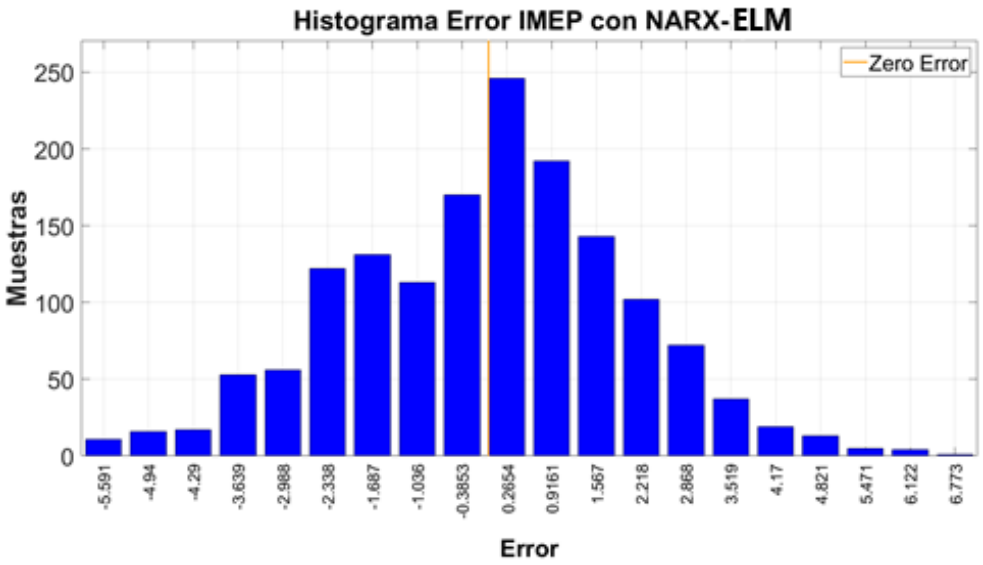

Figura 51. Histograma error IMEP con NARX-ELM

La media de la diferencia de valores del IMEP en la red NARX-ELM ha sido -0,0218 y la desviación típica de 2,0572.

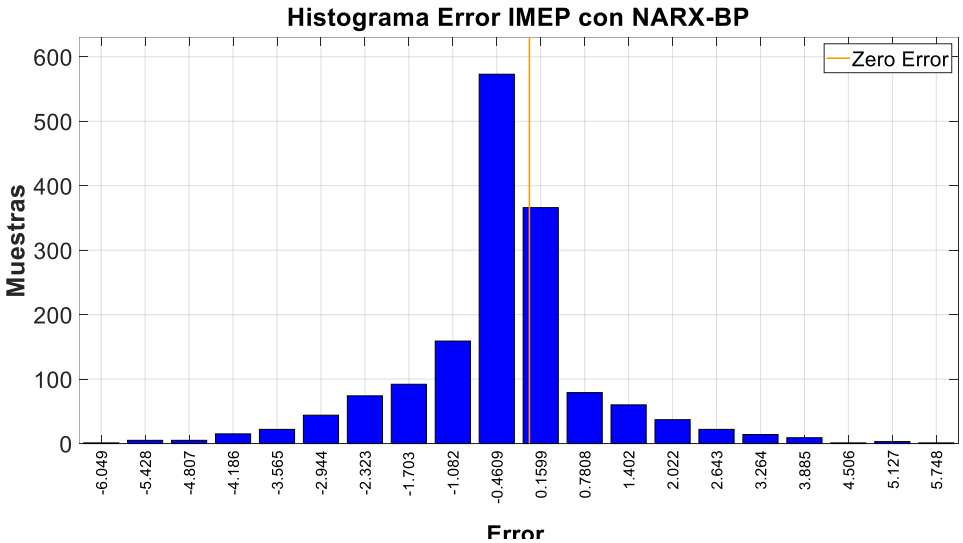

Figura 52. Histograma error IMEP con NARX-BP

La media de la diferencia de valores del IMEP en la red NARX-BP ha sido -0,3128 y la desviación típica de 1,3341.

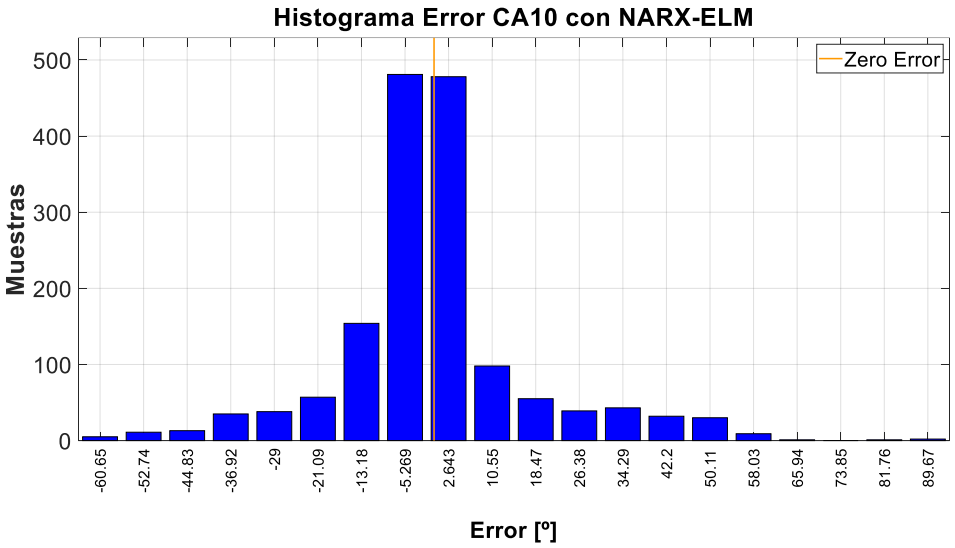

Figura 53. Histograma error CA10 con NARX-ELM

La media de la diferencia de valores del ca10 en la red NARX-ELM ha sido -0,4349 y la desviación típica de 17,9373.

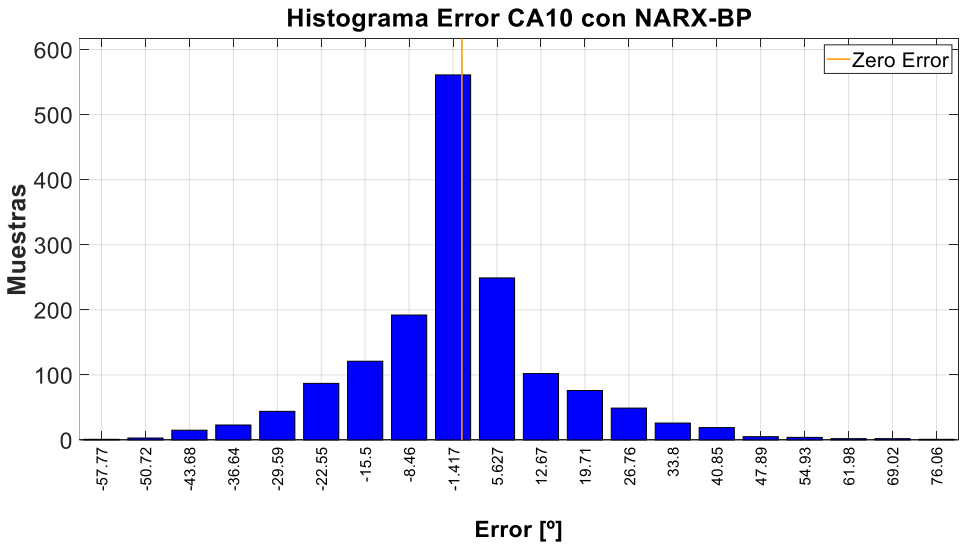

Figura 54. Histograma error CA10 con NARX-BP

La media de la diferencia de valores del ca10 en la red NARX-BP ha sido 0,0342 y la desviación típica de 15,3702.

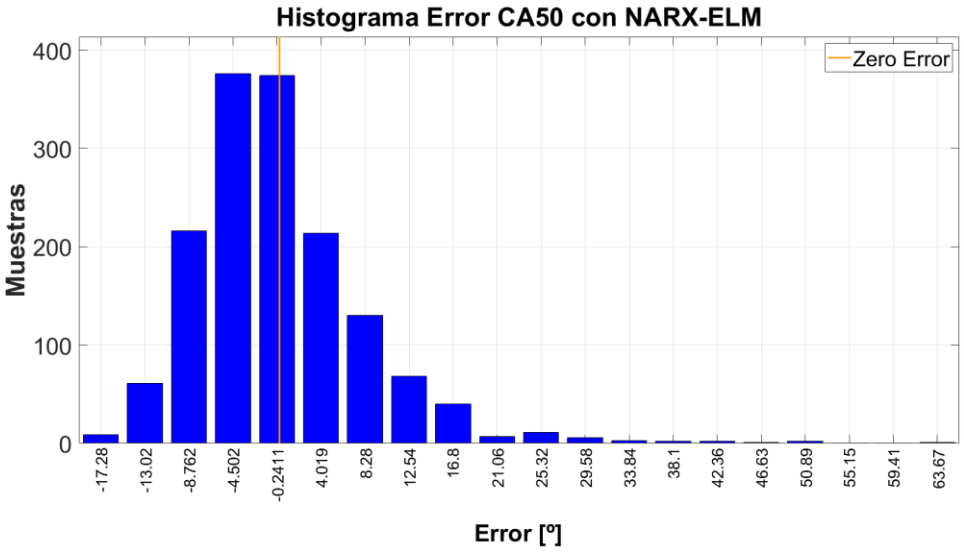

Figura 55. Histograma error CA50 con NARX-ELM

La media de la diferencia de valores del ca50 en la red NARX-ELM ha sido -0,0688 y la desviación típica de 8,4399.

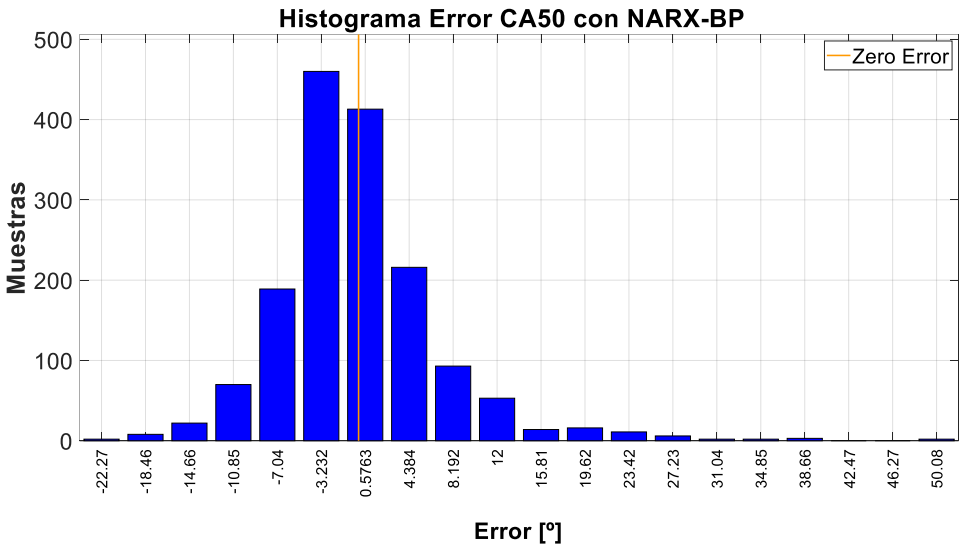

Figura 56. Histograma error CA50 con NARX-BP

La media de la diferencia de valores del ca50 en la red NARX-BP ha sido -0,1801 y la desviación típica de 7,2308.

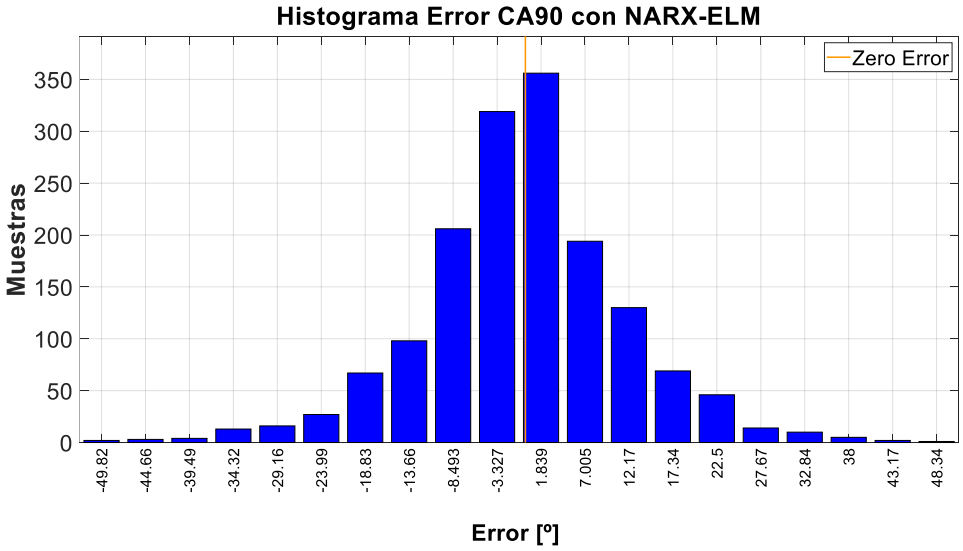

Figura 57. Histograma error CA90 con NARX-ELM

La media de la diferencia de valores del ca90 en la red NARX-ELM ha sido -0,3079 y la desviación típica de 11,9849.

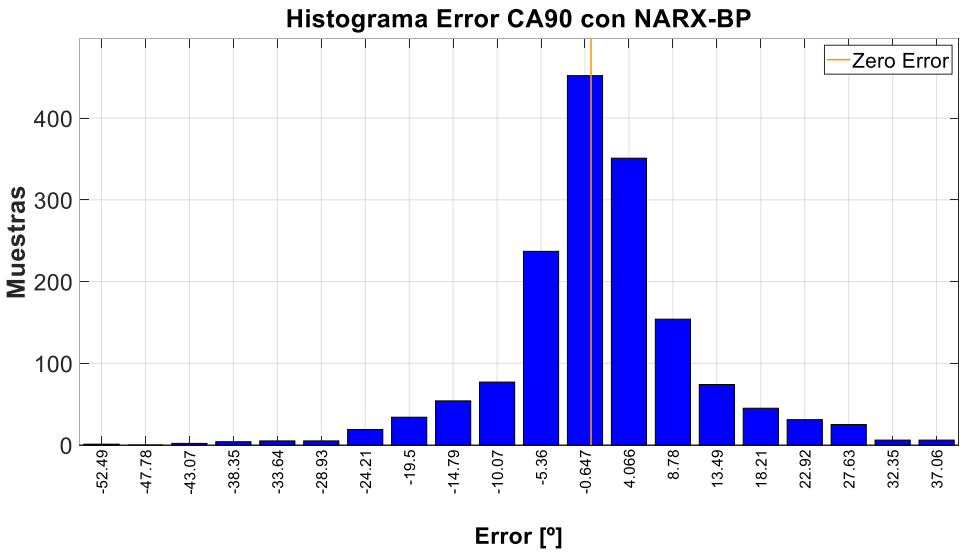

Figura 58. Histograma error CA90 con NARX-BP

La media de la diferencia de valores del ca90 en la red NARX-BP ha sido 0,5839 y la desviación típica de 9,2467.

Al igual que en el apartado anterior, es interesante realizar una tabla a modo de resumen, para mostrar los datos de manera que resulte sencillo extraer información. En ella también se van a exponer los datos obtenidos para cada parámetro mediante las redes *feed-forward.* Ya se ha comentado que con este tipo, a la hora de predecir modelos dinámicos, los errores adoptados son elevados ya que no consiguen adaptarse bien a este tipo de ensayos. Aun así es interesante expresarlos en las tablas que se suceden a continuación, para ratificar este hecho.

IMEP

	Mse [%]	Media	Desviación típica
ELM	9,8436		
MLP-BP	6,7140		
NARX-ELM	4,2299	-0,0218	2,0572
NARX-BP	1,9491	-0,3128	1,3341

Tabla 8. Resumen IMEP Dinámico

CA10

	Mse [%]	Media	Desviación típica
ELM	17,3498		
MLP-BP	15,8374		
NARX-ELM	8,2956	-0,4349	17,9373
NARX-BP	6,6471	0,0342	15,3702

Tabla 9. Resumen CA10 Dinámico

CA50

	Mse [%]	Media	Desviación típica
ELM	14,9081		
MLP-BP	13,8133		
NARX-ELM	9,3295	-0,0688	8,4399
NARX-BP	7,7104	-0,1801	7,2308

Tabla 10. Resumen CA50 Dinámico

CA90

	Mse [%]	Media	Desviación típica
ELM	15,1759		
MLP-BP	11,4703		
NARX-ELM	7,1280	-0,3079	11,9849
NARX-BP	5,3745	0,5839	9,2467

Tabla 11. Resumen CA90 Dinámico

Tiempo de entrenamiento

	Tiempo [s]
ELM	10,18
MLP-BP	329,62
NARX-ELM	11,27
NARX-BP	542,05

Tabla 12. Tiempo entrenamiento Dinámico

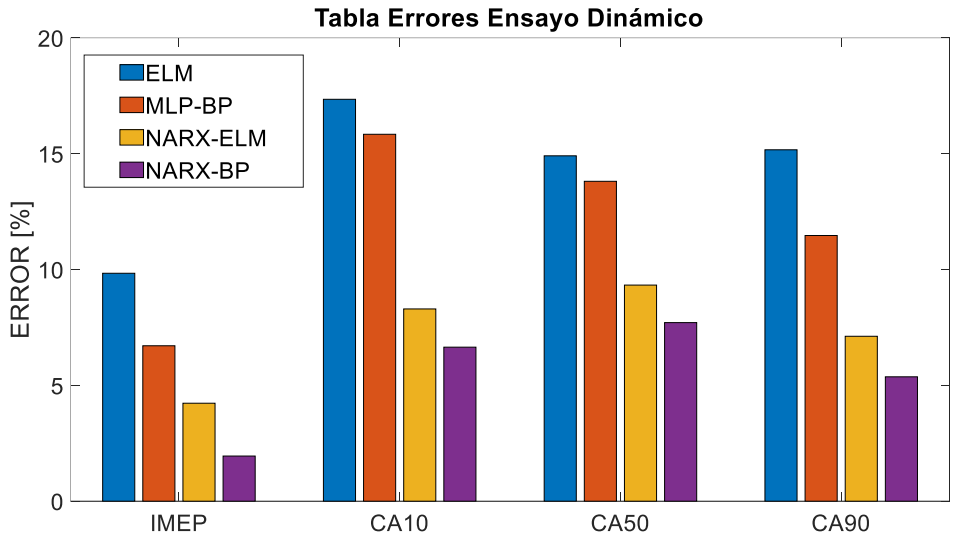

Figura 59. Diagrama errores Dinámico

5. CONCLUSIONES

Durante la explicación de los métodos utilizados para configurar la diferentes redes presentes en este trabajo se ha hecho especial hincapié en el hecho de que el Extreme Machine Learning era un proceso con poco coste computacional y que el algoritmo de Backpropagation era un método de entrenamiento más preciso pero con el inconveniente de que el tiempo de entrenamiento era mucho mayor. Esto se ha podido observar claramente durante todo el apartado de Resultados, ya sea en el ensayo estático o en el dinámico.

Analizando ambos ensayos por separado, en el estático se puede observar que la diferencia entre los errores de cada una de las dos configuraciones adoptadas (la ELM y la MLP-BP) es apreciable, siendo de media entre el estudio de los cuatro parámetros de un 3,87%. Por ello, sí que sería rentable desde el punto de vista del tiempo utilizar la red con un entrenamiento por retropropagación, ya que el tiempo empleado es mayor pero sin llegar a ser elevado (17,9 veces mayor), y en cambio la reducción del error es significativo.

En el caso del ensayo dinámico es diferente. En las redes NARX propuestas, los errores de la basada en ELM y la entrenada con Backpropagation (BP) no distan tanto unos de otros, ya que haciendo la media de esa diferencia de todos los estudios realizados (IMEP, CA10, CA50 y CA90) se obtiene 1,72%. Esto quiere decir, que no merece la pena utilizar la red NARX-BP, ya que el aumento del tiempo necesario para entrenar (48,09 veces mayor), no se refleja en una disminución significativa del error obtenido.

Recapitulando, si se desea minimizar el error a toda costa, porque se requiere realizar una control exhaustivo de la combustión, es preferible utilizar siempre las redes basadas en el entrenamiento por retropropagación, ya que en todos los casos se obtiene un menor error. Pero si lo que se desea es encontrar un equilibrio entre el tiempo utilizado y el error obtenido, lo ideal sería utilizar una red feed-forward basada en Backpropagation para el estudio del comportamiento de la combustión en condiciones estáticas y una red NARX basada en ELM para el estudio de las dinámicas.

6. PRESUPUESTO

El objetivo de este apartado es realizar un desglose del presupuesto necesario para la realización de este trabajo. En él se va a detallar los costes de mano de hora, de los diferentes softwares utilizados y los recursos personales.

6.1 MANO DE OBRA

El tiempo correspondiente para la realización de este trabajo son 300 horas, ya que el Trabajo de Fin de Grado tiene un peso de 12 créditos ECTS (25 horas/ 1 crédito). Además, según el Colegio Oficial de Ingenieros de Valencia (COGITI Valencia) el sueldo de un ingeniero debería situarse en el rango de 48 a 63 €/hora. En este caso se va a escoger un valor de 55 €/hora.

Este presupuesto se puede ver desglosado según las tareas a realizar en la Tabla 11.

TAREA	TIEMPO (Horas)	COSTE/HORA (€)	IMPORTE (€)
Modelado y configuración de sistemas	170	55	9350
Obtención de resultados	80		4400
Redacción	50		2750
Subtotal			16500

Tabla 13. Coste de mano de obra

6.2 SOFTWARE

En este apartado se va a detallar los costes de las diferentes licencias informáticas necesarias para poder realizar este proyecto (Tabla 12). Estas son anuales, pero como el trabajo solo ha ocupado un fracción de ese total es necesario realizar el coste de amortización (Tabla 13).

LICENCIAS	IMPORTE (€)
MATLAB	800
Microsoft Office 365 Personal	69
Subtotal	869

Tabla 14. Costes de Licencias

Periodo Amortización	1 año
Tiempo Amortización	5 meses
Coste Licencias	869
Coste/Mes	72,42
Subtotal	362,10

Tabla 15. Amortización de Licencias

6.2 RECURSOS PERSONALES

Estos costes hacen referencia a los diferentes recursos de los que hace uso el ingeniero encargado de realizar el trabajo, los cuales son de uso individual. El desglose de estos se puede observar en la Tabla 14 y la amortización en la Tabla 15.

RECURSO	**IMPORTE (€)**
Ordenador	1200
Licencia Windows 10 PRO	259
Subtotal	1459

Tabla 16. Costes de Recursos Personales

Periodo Amortización	6 años
Tiempo Amortización	5 meses
Coste Licencias	1459
Coste/Mes	20,22
Subtotal	101,11

Tabla 17. Amortización Recursos Personales

6.3 PRESUPUESTO TOTAL

En este último apartado se recoge el presupuesto total con todos los impuestos pertinentes.

CONCEPTO	IMPORTE (€)
Coste de Mano de obra	16500
Coste de Licencias	362,10
Coste de Recursos Personales	101,11
Total Ejecución Material	**16963,21**
Beneficio Industrial (6%)	1017,79
Total con Beneficio Industrial	**17981**
IVA (21%)	3776,01
Total Presupuesto	**21757,01**

Tabla 18. Presupuesto total

Asciende el presupuesto de ejecución material a la expresada cantidad en EUROS: DIECISEIS MIL NOVECIENTOS SESENTAITRES CON VIENTIDÓS

Asciende el presupuesto con beneficio industrial a la expresada cantidad en EUROS: DIECISIETE MIL NOVECIENTOS OCHENTA Y UNO

Asciende el presupuesto total a la expresada cantidad en EUROS: VEINTIÚN MIL SETECIENTOS CINCUENTA Y SIETE CON UNO

ANEXOS

ANEXO 1: Código extracción datos

GENERACIÓN DATOS VALIDOS TEST

```
load('test4.mat');

  % figure
    ax1=subplot(4,1,1);
    plot(data.fueldur(:,1));hold on;grid on;box on
    title(['Fueldur'])
    xlim([0 2777])
    set(gca,'FontSize',12)
    set(gca, 'XTickLabel', {''})

    ax2=subplot(4,1,2);
    plot(data.SA(:,1));hold on;grid on;box on
    title(['SA'])
    xlim([0 2777])
    set(gca,'FontSize',12)
    set(gca, 'XTickLabel', {''})

    ax3=subplot(4,1,3);
    plot(data.VGTdes(:,1));hold on;grid on;box on
    title(['THRdes'])
    xlim([0 2777])
    set(gca,'FontSize',12)
    set(gca, 'XTickLabel', {''})

    ax4=subplot(4,1,4);
    plot(data.IMEP(:,1));hold on;grid on;box on
    title(['IMEP'])
    xlim([0 2777])
    set(gca,'FontSize',12)
    set(gca, 'XTickLabel', {''})
```

EXTACCION VALORES VALIDOS TODOS LOS PARAMETROS

```
load('Intervalos2000.mat');
%Matriz intervalos extraida observando las zonas de combustion completa de todo el
ensayo
mediasV2000=zeros(size(Intervalos2000,1),10);

for j=1:size(Intervalos2000,1)
a=Intervalos2000(j,4);
z=(a-1);
b=Intervalos2000(j,5);
q=b-a;
s=zeros(q,10);
for i=a:b
   s(i-z,:)=[data.pint(i,:) data.maxp(i,:) data.maxdp(i,:) data.EGRpos(i,:)
data.mair(i,:) data.IMEP(i,:) data.CA10(i,:) data.CA50(i,:) data.CA90(i,:) data.t(i,:)];
end
```

```
mediasV2000(j,:)=mean(s);
end
```

CONJUNTO 3 TEST

```
load('fueldur1350.mat');
load('fueldur2000.mat');
load('fueldur3000.mat');
load('SA1350.mat');
load('SA2000.mat');
load('SA3000.mat');
load('THRdes1350.mat');
load('THRdes2000.mat');
load('THRdes3000.mat');
load('mediasV1350.mat');
load('mediasV2000.mat');
load('mediasV3000.mat');

fueldur=[fueldur1350; fueldur2000; fueldur3000];
SA=[SA1350; SA2000; SA3000];
THRdes=[THRdes1350; THRdes2000; THRdes3000];
mediasV=[mediasV; mediasV2000; mediasV3000];

R1=ones(36,1);
R2=ones(78,1);
R3=ones(75,1);

RPM=[R1*1350; R2*2000; R3*3000];
```

ANEXO 2: Red neuronal ELM

```
clear all;

load('fueldur.mat');
load('mediasV.mat');
load('SA.mat');
load('THRdes.mat');
load('RPM.mat');

msize = numel(SA);                              % Complete samples on the original test
td = randperm(msize, round(0.6*msize));       % Random index with 60% of the complete
sample

pint=mediasV(:,1);
mair=mediasV(:,5);
ca50=mediasV(:,9);

train.fd=fueldur(td);
train.SA =SA(td);
train.THR=THRdes(td);
train.ca50=ca50(td);
train.RPM=RPM(td);
train.pint=pint(td);
```

```
train.mair=mair(td);

X = [train.fd train.SA train.THR train.RPM train.pint train.mair];
y = [train. IMEP train.ca10 train.ca50 train.ca90];
```

```
X=(X-min(X))./range(X);
% y=(y-min(y))./range(y);


% mu = mean(X);                                % Media X
% sigma = std(X);                              % Desviacion estandar X
% X = [ones(size(X,1),1) (X-mu)./sigma];

% Hidden Layer
L = 50;                      % nodes in the hidden layer
a = rand(L,size(X,2));       % Pesos  %size->te devuelve dim 2 (columnas) de X
b = rand(1,L);           % Bias   corr_imep=-1.2
z = X*a'+b;                  % La ' es la transpuesta  %b [1x50] se suma a todas las
columnas
U = ones(size(z,1),L);       % Matriz unos (dim z)
H = U./(U + exp(-z));        % sigmoid activation
M = pinv(H);

%Output layer
beta = M*y;            % Moore-Penrose Pseudoinverse

%RELM (ELM with regularization)
lambda2 = 10;
n=size(H,2);
I = eye(n);
beta2 = (H'*H+I*lambda2)\H'*y;
```

```
valid.fd=fueldur; valid.fd(td)=[];
valid.SA =SA;  valid.SA(td)=[];
valid.THR=THRdes; valid.THR(td)=[];
valid.ca50=ca50; valid.ca50(td)=[];
valid.RPM=RPM; valid.RPM(td)=[];
valid.pint=pint; valid.pint(td)=[];
valid.mair=mair; valid.mair(td)=[];




Xv = [valid.fd valid.SA valid.THR valid.RPM valid.pint valid.mair];
Xv=(Xv-min(Xv))./range(Xv);
% Xv = [ones(size(Xv,1),1) (Xv-mu)./sigma];

yv = [valid.ca50];

zv = Xv*a'+b;
U2 = ones(size(zv,1),L);                % matriz unos (dim zv)
```

```
hv = U2./(U2 + exp(-zv));               % sigmoid activation

est_y=hv*beta;

error = zeros(length(Xv),4);
mse = zeros(length(Xv),4);
 for i = 1
%   error(:,i) = 100*(abs(yv(:,i)-est_y(:,i))./yv(:,i));       % Relative  error imep
    error(:,i) = 100*(abs(yv(:,i)-est_y(:,i))./range(yv));       % Relative error ca's
    mse(:,i) = mean((yv(:,i)-est_y(:,i)).^2);                  % Mean squared error
 end
%  yv=yv*1e-5;
%  est_y=(est_y*1e-5)+1;
```

```
%
% yv(:,1)=yv(:,1)*1e-5;
% est_y(:,1)=est_y(:,1)*1e-5;
% mse(:,1) = mean((yv(:,1)-est_y(:,1)).^2);

    figure
    ax1=subplot(1,1,1);
    plot(yv(:,1));hold on;grid on;box on
    plot(est_y(:,1),'k')
    title(['Estimación IMEP'])
    ylabel('IMEP [Bar]')
    ylim([-100 100]);xlim([0 1582])
    legend('Observado','Predicho','Location','SE')
    set(gca,'FontSize',30)
    set(gca, 'XTickLabel', {''})


 syms x;
p1= x;
    figure
    ax1=subplot(1,1,1);
    plot(est_y(:,1),yv(:,1),'.','markersize',20);hold on;grid on;box on
    fplot(x,'g')
    title(['Estimación IMEP'])
    ylabel('MEDIDO [Bar]');  xlabel('PREDICHO [Bar]')
    axis([min(est_y(:,1)) max(est_y(:,1)) min(yv(:,1)) max(yv(:,1))])
    set(gca,'FontSize',30)
    set(gca, 'XTickLabel', {''})
%
    figure
    ax1=subplot(1,1,1);
    plot(est_y(:,2),yv(:,2),'.','markersize',20);hold on;grid on;box on
    fplot(x,'g')
    title(['Estimación CA10'])
    ylabel('MEDIDO [º]');  xlabel('PREDICHO [º]')
    axis([min(est_y(:,2)) max(est_y(:,2)) min(yv(:,2)) max(yv(:,2))])
    set(gca,'FontSize',30)
    set(gca, 'XTickLabel', {''})

    figure
    ax1=subplot(1,1,1);
```

```
    plot(est_y(:,3),yv(:,3),'.','markersize',20);hold on;grid on;box on
    fplot(x,'g')
    title(['Estimación CA50'])
    ylabel('MEDIDO [º]');  xlabel('PREDICHO [º]')
    axis([min(est_y(:,3)) max(est_y(:,3)) min(yv(:,3)) max(yv(:,3))])
    set(gca,'FontSize',30)
    set(gca, 'XTickLabel', {''})

    figure
    ax1=subplot(1,1,1);
    plot(est_y(:,4),yv(:,4),'.','markersize',20);hold on;grid on;box on
    fplot(x,'g')
    title(['Estimación CA90'])
    ylabel('MEDIDO [º]');  xlabel('PREDICHO [º]')
    axis([min(est_y(:,4)) max(est_y(:,4)) min(yv(:,4)) max(yv(:,4))])
    set(gca,'FontSize',30)
    set(gca, 'XTickLabel', {''})



mse(1,1)
% mse(1,2)
% mse(1,3)
% mse(1,4)
```

ANEXO 3: Red Neuronal MLP con Backpropagation

```
clear all;

load('fueldur.mat');
load('mediasV.mat');
load('SA.mat');
load('THRdes.mat');
load('RPM.mat');


msize = numel(SA);                              % Complete samples on the original test
td = randperm(msize, round(0.6*msize));       % Random index with 75% of the complete
sample

pint=mediasV(:,1);
mair=mediasV(:,5);
ca10=mediasV(:,9);


train.fd=fueldur(td);
train.SA =SA(td);
train.THR=THRdes(td);
train.ca10=ca10(td);
train.RPM=RPM(td);
```

```
train.pint=pint(td);
train.mair=mair(td);


X = [train.fd train.SA train.THR train.RPM train.pint train.mair];
y = [train. IMEP train.ca10 train.ca50 train.ca90];
```

```
hidden_layers=[50];
% scales data between 0 and 1
Xa=(X-min(X))./range(X);% adimensionalize between 0 and 1
ya=(y-min(y))./range(y);% adimensionalize between 0 and 1
network_architecture=[(size(Xa,2)) hidden_layers (size(y,2))];
Number_W=length(network_architecture)-1;
% generate initial weights estimate
layer=struct('W',0,'value',0,'der',0,'trans',0);% initialize hidden
for i=1:Number_W
   layer(i).W=rand(network_architecture(i),network_architecture(i+1))-0.5;%generate
weights
end
```

```
max_it=1e5;
n_train=size(X,1);
k_ac=10/n_train;
err_all=ones(max_it,1);
tot=1:1:size(X,1);
tot_train = tot(randperm(n_train));
it=1;
out_m=zeros(size(X,1),size(y,2));
```

```
while (err_all(it)>=0.05&&it<=max_it)
        it=it+1;
%     for sa=1:n_train                        % number of instances
        input=Xa(tot_train,:);                % taking one instance each time
        in_la=input;                          % input at each layer
        output=ya(tot_train,:);               % load one target each time

        % b. calculate hidden layers (forward propagation)
            for i=1:Number_W
                Wi=layer(i).W;
                layer(i).value=sigmoid(in_la*Wi); % calculate the hidden layer
                layer(i).der=layer(i).value.*(1-layer(i).value);
                if i==Number_W
                  out_m=layer(i).value;
                else
                  in_la=layer(i).value;            % set the actual hidden layers as the
input of the next layer
                end
            end
        % c. calculate the error
        errorTrain=output-out_m;
        err_all(it)=sqrt(mse(errorTrain));
        % d.update weights if the error doesn't satisfy our coditions (C2)
        % d.1. calculate traversing_value of each  hidden layer (backword propagation)

        %%%% backpropagation starts from here%%%%
         for i=Number_W:-1:1
```

```
            if i==Number_W
                layer(i).trans=errorTrain.*layer(i).der; % compute & Store The
traversing_value
            else
               layer(i).trans=layer(i).der.*(layer(i+1).trans*layer(i+1).W');
            end
        end
        %%%% backpropagation ends here%%%%
        % d.2.weights correction
        for i=1:Number_W
               if i==1
                layer(i).W=layer(i).W+k_ac*(input'*layer(i).trans); % update weights
step1
               else
                layer(i).W=layer(i).W+k_ac*(layer(i-1).value'*layer(i).trans); % update
weights step1
               end
        end
        if mod(it,100)==0
            fprintf('iteración %d, error %.2f \n',it, err_all(it)*100)
        end

end
```

```
valid.fd=fueldur; valid.fd(td)=[];
valid.SA =SA;  valid.SA(td)=[];
valid.THR=THRdes; valid.THR(td)=[];
valid.ca10=ca10; valid.ca10(td)=[];
valid.RPM=RPM; valid.RPM(td)=[];
valid.pint=pint; valid.pint(td)=[];
valid.mair=mair; valid.mair(td)=[];



Xv = [valid.fd valid.SA valid.THR valid.RPM valid.pint valid.mair];
Xval=(Xv-min(Xv))./range(Xv);
% Xv = [ones(size(Xv,1),1) (Xv-mu)./sigma];

yv = [valid.ca10];

yval=(yv-min(yv))./range(yv);        % adimensionalize between 0 and 1

n_valid=size(Xv,1);
totV=1:1:size(Xv,1);
% tot_valid = totV(randperm(n_valid));

out_all=ones(n_valid,4);
for sa=1:totV                          % number of instances
        input= Xval(totV,:);                        % taking one instance each time
        in_la= input;                          % input at each layer
        output= yval(totV,:);              % load one target each time

%         b. calculate hiddenlayers (forward propagation)
           for i=1:Number_W
               Wi=layer(i).W;
```

```
                if i==Number_W
                  out_all=sigmoid(in_la*Wi);
                else
                  in_la=sigmoid(in_la*Wi);            % set the actual hidden layers as
the input of the next layer
                end
            end
end

output_dim =(output.*range(y))+min(y);
outall_dim =(out_all.*range(y))+min(y);



for i = 1
    errorValid(:,i) = output_dim(:,i)-outall_dim(:,i);
%   errorValid2(:,i) = 100*abs(output_dim(:,i)-outall_dim(:,i))./output_dim(:,i);   %
Relative error
    errorValid2(:,i) = 100*abs(output_dim(:,i)-outall_dim(:,i))./range(yv);   % Relative
error
    Mse(:,i)= sqrt(mse(errorValid));                                          % Mean squared
error
end
```

```
syms x;
p1= x;
    figure
    ax1=subplot(1,1,1);
    plot(outall_dim(:,1),output_dim(:,1),'.','markersize',20);hold on;grid on;box on
    fplot(x,'g')
    title(['IMEP estimation'])
    ylabel('MEDIDO [º]');  xlabel('PREDICHO [º]')
    axis([min(outall_dim) max(outall_dim) min(outall_dim) max(outall_dim)])
    set(gca,'FontSize',30)
    set(gca, 'XTickLabel', {''})
```

ANEXO 4: Red neuronal NARX con ELM

```
clear all;

load('fueldur.mat');
load('mediasV.mat');
load('SA.mat');
load('THRdes.mat');
load('RPM.mat');


msize = numel(SA);                              % Complete samples on the original test
td = randperm(msize, round(0.6*msize));       % Random index with 75% of the complete
sample

pint=mediasV(:,1);
mair=mediasV(:,5);
ca10=mediasV(:,9);


train.fd=fueldur(td);
train.SA =SA(td);
train.THR=THRdes(td);
train.ca10=ca10(td);
train.RPM=RPM(td);
train.pint=pint(td);
train.mair=mair(td);


X = [train.fd train.SA train.THR train.RPM train.pint train.mair];
y = [train.IMEP train.ca10 train.ca50 train.ca90];
% Establish input and output parameters

Xpasado = zeros(size(X));
for i = 1:(size(X,1)-1)
    Xpasado(i+1,:)=X(i,:);
end
X = [train.fd train.SA train.THR train.pint train.mair Xpasado];

X=(X-min(X))./range(X);% adimensionalize between 0 and 1


% Hidden Layer
tic
L = 50;                     % nodes in the hidden layer
a = rand(L,size(X,2));      % Pesos  %size->te devuelve dim 2 (columnas) de X
b = rand(1,L)+2;              % Bias
z = X*a'+b;                 % La ' es la transpuesta  %b [1x50] se suma a todas las
columnas
U = ones(size(z,1),L);      % Matriz unos (dim z)
H = U./(U + exp(-z));       % sigmoid activation

%Output layer
beta = pinv(H)*y;           % Moore-Penrose Pseudoinverse
elpased_time = toc;

%RELM (ELM with regularization)
lambda2 = 0.5;
```

```
n=size(H,2);
I = eye(n);
beta2 = (H'*H+I*lambda2)\H'*y;
```

```
valid.fd=fueldur; valid.fd(td)=[];
valid.SA =SA;  valid.SA(td)=[];
valid.THR=THRdes; valid.THR(td)=[];
valid.ca10=ca10; valid.ca10(td)=[];
valid.RPM=RPM; valid.RPM(td)=[];
valid.pint=pint; valid.pint(td)=[];
valid.mair=mair; valid.mair(td)=[];



Xv = [valid.fd valid.SA valid.THR valid.RPM valid.pint valid.mair];
yv = [valid.ca10];

XpasadoV = zeros(size(Xv));

for i = 1:(size(Xv,1)-1)
    XpasadoV(i+1,:)=Xv(i,:);
end
Xv = [valid.fd valid.SA valid.THR valid.pint valid.mair XpasadoV];
Xv =(Xv-min(Xv))./range(Xv);



zv = Xv*a'+b;
U2 = ones(size(zv,1),L);                % matriz unos (dim zv)
hv = U2./(U2 + exp(-zv));               % sigmoid activation

est_y=hv*beta;

error = zeros(length(Xv),4);
mse = zeros(1,4);
for i = 1
    error(:,i) = 100*(abs(yv(:,i)-est_y(:,i))./yv(:,i));       % Relative error
    error(:,i) = 100*(abs(yv(:,i)-est_y(:,i))./range(yv));       % Relative error ca's
    mse(:,i) = mean((yv(:,i)-est_y(:,i)).^2);                % Mean squared error
end
```

```
% %% FIGURES

yv=yv*1e-5;
est_y=est_y*1e-5;
est_y=est_y(60:1582,:);
yv=yv(60:1582,:);
errorI=errorI(3:1582,:);
errorI=errorI*1e-5;
errorC=errorC(3:1582,:);
mseX(:,1) = mean((yv(:,1)-est_y(:,1)).^2);
mseX(1,1)

DiferenciaELM_IMEP= yv(:,1)-est_y(:,1);
```

```
    figure
     ax1=subplot(1,1,1);
     plot(yv(:,1));hold on;grid on;box on
     plot(est_y(:,1),'k')
     title(['Estimación IMEP'])
     ylabel('IMEP [º]')
     ylim([-50 200]);xlim([0 1523])
     legend('Observed','Predicted','Location','SE')
     set(gca,'FontSize',30)
     set(gca, 'XTickLabel', {''})

%      ax2=subplot(2,1,2);
%      plot(errorC(:,1));grid on;box on
%      xlim([0 1523]);
%      xlabel('cycles')
%      ylabel('Error [%]')
%      set(gca,'FontSize',11)
%      linkaxes([ax1 ax2],'x')

%      mse(1,1)
```

ANEXO 5: Red neuronal NARX con Backpropagation

```
clear all;

load('fueldur.mat');
load('mediasV.mat');
load('SA.mat');
load('THRdes.mat');
load('RPM.mat');


msize = numel(SA);                              % Complete samples on the original test
td = randperm(msize, round(0.6*msize));       % Random index with 75% of the complete
sample

pint=mediasV(:,1);
mair=mediasV(:,5);
ca10=mediasV(:,7);


train.fd=fueldur(td);
train.SA =SA(td);
train.THR=THRdes(td);
train.ca10=ca10(td);
train.RPM=RPM(td);
train.pint=pint(td);
train.mair=mair(td);


X = [train.fd train.SA train.THR train.RPM train.pint train.mair];
y = [train.IMEP train.ca10 train.ca50 train.ca90];

% Establish input and output parameters

Xpasado = zeros(size(X));
for i = 1:(size(X,1)-1)
    Xpasado(i+1,:)=X(i,:);
end
Ypasado = zeros(size(y));
for i = 1:(size(y,1)-1)
    Ypasado(i+1,:)=y(i,:);
end
X = [train.fd train.SA train.THR train.pint train.mair Xpasado Ypasado];

X=(X-min(X))./range(X);% adimensionalize between 0 and 1
```

```
hidden_layers=[50];
% scales data between 0 and 1
Xa=(X-min(X))./range(X);% adimensionalize between 0 and 1
ya=(y-min(y))./range(y);% adimensionalize between 0 and 1
network_architecture=[(size(Xa,2)) hidden_layers (size(y,2))];
Number_W=length(network_architecture)-1;
% generate initial weights estimate
layer=struct('W',0,'value',0,'der',0,'trans',0);% initialize hidden
for i=1:Number_W
   layer(i).W=rand(network_architecture(i),network_architecture(i+1))-0.5;%generate
```

```
weights
end
```

```
max_it=1e5;
n_train=size(X,1);
k_ac=10/n_train;
err_all=ones(max_it,1);
tot=1:1:size(X,1);
tot_train = tot(randperm(n_train));
it=1;
out_ent=ones(n_train,size(ya,2));
```

```
while (err_all(it)>=0.06&&it<=max_it)
        it=it+1;
%     for sa=1:n_train% number of instances
        out_m=out_ent;
        input=[Xa(tot_train,:)];               % taking one instance each time
out_m(tot_train,:)
        in_la=input;                                                % input at each layer
        output=ya(tot_train,:);              % load one target each time

        % b. calculate hidden layers (forward propagation)
            for i=1:Number_W
                Wi=layer(i).W;
                layer(i).value=sigmoid(in_la*Wi); % calculate the hidden layer
                layer(i).der=layer(i).value.*(1-layer(i).value);
                if i==Number_W
                  out_m=layer(i).value;
                else
                  in_la=layer(i).value;            % set the actual hidden layers as the
input of the next layer
                end
            end
        % c. calculate the error
        errorTrain=output-out_m;
        err_all(it)=sqrt(mse(errorTrain));
        % d.update weights if the error doesn't satisfy our coditions (C2)
        % d.1. calculate traversing_value of each  hidden layer (backword propagation)

        %%%% backpropagation starts from here%%%%
         for i=Number_W:-1:1
             if i==Number_W
                 layer(i).trans=errorTrain.*layer(i).der; % compute & Store The
traversing_value
             else
                layer(i).trans=layer(i).der.*(layer(i+1).trans*layer(i+1).W');
             end
         end
         %%%% backpropagation ends here%%%%
         % d.2.weights correction
         for i=1:Number_W
                if i==1
                 layer(i).W=layer(i).W+k_ac*(input'*layer(i).trans); % update weights
step1
                else
```

```
                layer(i).W=layer(i).W+k_ac*(layer(i-1).value'*layer(i).trans); % update
weights step1
            end
        end
        if mod(it,100)==0
            fprintf('iteración %d, error %.2f \n',it, err_all(it)*100)
        end
end
```

```
valid.fd=fueldur; valid.fd(td)=[];
valid.SA =SA;  valid.SA(td)=[];
valid.THR=THRdes; valid.THR(td)=[];
valid.ca10=ca10; valid.ca10(td)=[];
valid.RPM=RPM; valid.RPM(td)=[];
valid.pint=pint; valid.pint(td)=[];
valid.mair=mair; valid.mair(td)=[];



Xv = [valid.fd valid.SA valid.THR valid.RPM valid.pint valid.mair];
yv = [valid.ca10 valid.ca10 valid.ca0];

XpasadoV = zeros(size(Xv));

for i = 1:(size(Xv,1)-1)
    XpasadoV(i+1,:)=Xv(i,:);
end
YpasadoV = zeros(size(yv));
for i = 1:(size(yv,1)-1)
    YpasadoV(i+1,:)=y(i,:);
end
Xv = [valid.fd valid.SA valid.THR valid.pint valid.mair XpasadoV YpasadoV];
Xv =(Xv-min(Xv))./range(Xv);

Xval=(Xv-min(Xv))./range(Xv);                                   % adimensionalize between 0
and 1
yval=(valid.ca10-min(valid.ca10))./range(valid.ca10);        % adimensionalize between 0
and 1
n_valid=size(Xv,1);
totV=1:1:size(Xv,1);
% tot_valid = totV(randperm(n_valid));
out_ent2=ones(n_valid,1);
out_all=ones(n_valid,1);
for sa=1:totV                          % number of instances
        out_all=out_ent2;
        input= [Xval(totV,:)];                          % taking one instance each time
out_all(totV,:)
        in_la= input;                           % input at each layer
        output= yval(totV,:);              % load one target each time

%         b. calculate hiddenlayers (forward propagation)
          for i=1:Number_W
              Wi=layer(i).W;
              if i==Number_W
                out_all=sigmoid(in_la*Wi);
```

```
                else
                  in_la=sigmoid(in_la*Wi);            % set the actual hidden layers as
the input of the next layer
                end
            end
end

output_dim =(output.*range(valid.ca10))+min(valid.ca10);
outall_dim =(out_all.*range(valid.ca10))+min(valid.ca10);

% outall_dim=outall_dim+3;

    errorValid(:,1) = output_dim(:,1)-outall_dim(:,1);
%     errorValid2(:,1) = 100*abs(output_dim(:,1)-outall_dim(:,1))./output_dim(:,1);   %
Relative error
    errorValid2(:,1) = 100*abs(output_dim(:,1)-outall_dim(:,1))./range(yv);   % Relative
error

    Mse(:,1)= sqrt(mse(errorValid));                                          % Mean squared
error
```

```
% output_dim=output_dim/1e5;
% outall_dim=outall_dim/1e5;

    figure
    ax1=subplot(1,1,1);
    plot(output_dim(:,1));hold on;grid on;box on
    plot(outall_dim(:,1),'k')
    title(['Estimación CA90'])
    ylabel('CA90 [º]')
    ylim([-50 200]); xlim([0 1582])
    legend('Observed','Predicted','Location','SE')
    set(gca,'FontSize',30)
    set(gca, 'XTickLabel', {''})
```

BIBLIOGRAFÍA

[1] Bengio, Y. (2009, enero 01). "*Learning Deep Architectures for AI*". Recuperado de: https://www.iro.umontreal.ca/~lisa/pointeurs/TR1312.pdf

[2] Izaurieta, F., Saavedra, C. (2000, septiembre 08). *"Redes Neuronales Artificiales".* Recuperado de: http://www.franciscocruz.cl/lectures/sistint/rna.pdf

[3] Amrutha, J., Remya Ajai, A. S. (2018, mayo 18). *"Performance analysis of Backpropagation Algorithm of Artificial Neural Networks in Verilog".* Recuperado de: https://ieeexplore.ieee.org/abstract/document/9012614

[4] Cocco Mariani, V., Hennings Och, S., dos Santos Coelho, L., Domingues, E. (2019, septiembre 01). *"Pressure prediction of a spark ignition single cylinder engine using optimized extreme learning machine models".* Recuperado de: https://www.sciencedirect.com/science/article/abs/pii/S0306261919307974

[5] Boussaada, Z., Curea, O., Remaci, A., Camblong, H., Mrabet Bellaaj, N. (2018, marzo 10). *"A Nonlinear Autoregressive Exogenous (NARX) Neural Network Model for the Prediction of the Daily Direct Solar Radiation".* Recuperado de: https://www.mdpi.com/1996-1073/11/3/620/htm

[6] Omran, R., Younes, R., Champoussin, J., Outbib, R. (2011, septiembre 03). "New Indicated Mean Effective Pressure (IMEP) model for predicting crankshaft movement". Recuperado de: https://www.researchgate.net/publication/237202236_New_Indicated_Mean_Effective_Pressure_IMEP_model_for_predicting_crankshaft_movement

[7] MAHLE Powertrain. *"MAHLE jet Ignition".* Recuperado de: https://www.mahle-powertrain.com/en/experience/mahle-jet-ignition/