UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UN MODELO BASADO EN REDES NEURONALES ADAPTATIVAS PARA OPTIMIZAR EL FUNCIONAMIENTO DE LA NUEVA GENERACIÓN DE MCIA

AUTOR: BOSCH ALONSO, VICENTE

TUTOR: NOVELLA ROSA, RICARDO

COTUTOR: BARES MORENO, PAU

Curso Académico: 2020-21

Agradecimientos

Mi más sincero y firme agradecimiento a mi director del trabajo, Pau Bares, el cual nunca dejó de guiarme en el desarrollo de este trabajo, sin decirme más de lo que necesitaba para que fuera yo el que entendiera y llegara a las conclusiones adecuadas, dándome cuenta de mis propios errores; gracias también por su apoyo en el trabajo, sobre todo durante las últimas semanas. Por todo esto y más, gracias Pau. Por otra parte, debo agradecer el apoyo de Alexandra Aramburu, estudiante de doctorado en el CMT, que me proporcionó todos los datos necesarios y que no dudó en prestarme ayuda con cualquier duda relacionada con estos.

Tampoco puedo olvidarme de todos los compañeros y amigos que me han acompañado durante todo el grado, es imposible contar con las manos a toda esta gente, y mucho menos posible contar los buenos momentos que pasé con ellos, desde el primer al último día, sin que hubiera ningún día que no disfrutara de su compañía. Por haberlos conocido les estaré siempre agradecido.

Infinitas gracias a mi familia, en especial a mis padres, quienes estuvieron a mi lado en mis mejores y peores días, sin pedirme en ningún momento más responsabilidades de las necesarias cuando más apretaba el tiempo de estudio, por darme la oportunidad de estudiar aquello que me apasiona, sin vosotros no sería nada de lo que soy ni estaría dónde estoy. Por esto y por mucho más, gracias.

Resumen

En este trabajo se plantea un modelo basado en redes neuronales supervisadas adaptativas para identificar y predecir el comportamiento de distintos parámetros de un motor térmico, realizando el desarrollo, entrenamiento de la red neuronal y validación del modelo mediante el programa MATLAB.

El algoritmo propuesto puede ser ejecutado en tiempo real y permite la adaptación del sistema, permitiendo así predecir el comportamiento del motor, incluso cuando ocurren fallos, envejecimiento o dispersión en la fabricación.

Palabras Clave: Redes neuronales adaptativas; machine learning; inteligencia artificial; optimización; motor de combustión interna alternativo.

Abstract

It this work, a model based on adaptative supervised neuronal networks is proposed to identify and predict the behaviour of different parameters on a reciprocrating engine, performing the development, training of the neuronal network and validation of the model through MATLAB.

The proposed algorithm can be executed in real time and allows the adaptation of the system, predicting the behaviour of the engine, even if there are failures, aging or manufacturing dispersion.

Keywords: Adaptative neuronal networks; machine learning; artificial intelligence; optimization; reciprocating internal combustion engine.

Resum

En aquest treball es planteja un model basat en xarxes neuronals supervisades adaptatives per identificar i predir el comportament de distints paràmetres de un motor tèrmic, realitzant el desenvolupament, entrenament de la xarxa neuronal i validació del model mitjançant el programa MATLAB.

El algoritme proposat pot se executat en temps real i permet l'adaptació del sistema, permitent així predir el comportament del motor, inclòs quan ocorren falles, envelliment o dispersions a la fabricació.

Paraules clau: Xarxes neuronals adaptatives; Machine learning; intel·ligència artificial; optimització; motor de combustió interna alternatiu.

Índice general

Índice de figuras

Parte I

Memoria

Capítulo 1

Introducción

1.1 Motivación

Con una previsión del fin del petroleo en unos 50 años y un aumento exponencial de las emisiones de gases contaminantes, que durante el último siglo han crecido de manera exponencial, se ha desarrollado una conciencia global acerca de la reducción de emisiones, con el fin de cuidar el planeta en el que vivimos. Para ello es necesario un desarrollo sostenible, tanto de la sociedad como de la economía, y por tanto la industria; y el uso de unas prácticas que nos permitan mantener los ritmos de crecimientos económicos actuales pero reduciendo el gasto energético.

En el sector del transporte actual, en España, representa el 25 % de las emisiones de dióxido de carbono (CO_2), representando el transporte por carretera el 95 % de estas emisiones, como se puede consultar en la página web del Instituto de la Diversificación y Ahorro de la Energía (IDAE). Es por eso que cada vez se investiga más sobre nuevas tecnologías aplicadas a los motores de combustión que permitan reducir sus emisiones y aumentar su rendimiento. Algunas de estas nuevas tecnologías que se encuentran en desarrollo son el uso de una pre-cámara, o TJI, que se usa en motores de encendido provocado, o el uso de una válvula EGR de baja presión.

Por lo tanto, seguir haciendo el máximo esfuerzo en la dirección adecuada para la optimización de los motores de combustión, y por tanto la reducción de gases contaminantes producidos, debe ser una exigencia social forjada por el compromiso medio-ambiantel que nos une en estos momentos.

1.1.1 Introducción nuevos MCIA

Un motor de combustión interna alternativo (MCIA) se puede definir como un conjunto de elementos mecánicos que mediante un movimiento alternativo permite obtener energía mecánica a partir de la energía generada por un proceso de combustión en el seno del fluido que lo atraviesa. Esta definición es muy amplia, existiendo una gran variedad de motores en función de distintas características. El inicio de los motores tal y como los conocemos a día de hoy se dio en 1876, cuando el alemán Nikolas August Otto desarolló el motor basado en el ciclo de Otto. Posteriormente, Rudolf Diesel, con la intención de aproximar el ciclo de Otto al ciclo ideal de Carnot desarrolló el motor de Diesel. Estos dos motores son lo que hoy en día se conocen como los motores MEP (motor de encendido provocado) y MEC (motor de encendido por compresión).

Otra manera de divirdirlos es atendiendo al ciclo de trabajo, donde se pueden encontrar los motores de dos tiempos y los de cuatro tiempos, la diferencia entre estos es que en el motor de dos tiempos se realiza el proceso de combustión cada dos recorridos del pistón, mientras que para el motor de cuatro tiempos se necesitarán cuatro recorridos del pistón. Por otra parte, también se pueden clasificar los motores en función de la presión de admisión, existiendo motores de aspiración natural, dónde el aire en la admisión se encuentra a presión atmosférica, y motores sobrealimentados, en los que antes de introducirse el aire en la admisión han pasado por un proceso de compresión, bien por un compresor mecánico o bien por un turbocompresor; otra característica es el tipo de refrigeración, si bien es cierto que la gran mayoría de los motores se refrigeran con agua, existen motores de menor potencia que pueden refrigerarse con aire. Finalmente también se pueden clasificar los motores en función del número de cilindros y su disposición, existiendo motores en línea, en V, etc. A pesar de estas características existen muchas más, tanto cuantitativas como cualitativas que definen un motor.

A lo largo de este trabajo se va a trabajar con dos motores, por lo que se va a realizar una breve introducción de los motores en general, para después particularizar para cada uno de ellos. De estos motores se va a estudiar principalmente el proceso de combustión, por lo que es de gran interés conocer el lugar dónde ésta se va a producir.

Una de las principales partes de un motor, y mediante la cual se transforma la energía química del combustible en energía mecánica, está formada por la cámara de combustión y el mecanismo biela-manivela. Además hay elementos que permiten la renovación de la carga y la inyección del combustible.

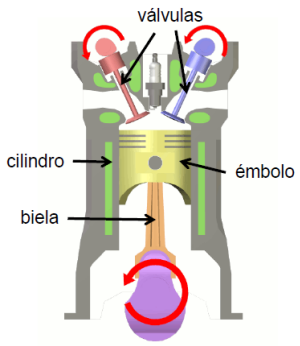


Figura 1.1: Esquema de cámara de combustión y biela-manivela. Fuente: Cruz y col., 2002

La cámara de combustión es el lugar en el que se produce la extracción de la energía química del combustible mediante el proceso de combustión. Esta cámara está generalmente refrigerada por agua mediante unos conductos en sus paredes que evitan un sobrecalentamiento de esta. Sobre la cámara de combustión se adhieren una serie de elementos que son los encargados de realizar la renovación de la carga. El primero de ellos son las válvulas de admisión y escape, estás válvulas permiten abrir y cerrar el acceso de los conductos de admisión y escape con la cámara de combustión; estas válvulas se abren de manera sincronizada con el movimiento del émbolo, aprovechando el movimiento de la biela para absorber o expulsar los gases, siendo la correcta apertura y cierre de estas válvulas una labor fundamental para una correcta combustión. Otro elemento es la bujía, así como las válvulas de admisión y escape siembre van a estar presentes en un motor, la bujía tan solo se encontrará en los motores de encendido provocado. La función de la bujía es la de aportar la energía necesaria para que se inicie la combustión en los motores cuyo combustible es la gasolina. Esta energía se aporta en forma de una chispa que se produce entre dos electrodos, de tal forma que en el lugar donde se produce la chispa es el lugar en el que se inicia la combustión. Otro elemento muy común, sobre todo en los motores de encendido por compresión, es el inyector. Existen distintas formas de inyectar el combustible en la cámara de combustión, el más sencillo de ellos es mezclarlo con el aire e inyectarlo mediante el conducto de admisión; el otro es mediante un inyector. El inyector es el encargado de introducir el combustible en motores con inyección directa, generalmente MEC; este es capaz de inyectar el combustible a una elevada presión con el fin de atomizar el combustible y tener mayor superficie efectiva para el proceso de combustión, este proceso de inyección puede ser controlado mediante el controlador del motor (ECU), permitiendo controlar cuando se inicia el proceso de combustión; esta tecnología que permite inyectar el combustible a alta presión es conocido como *common rail*.

La otra parte de interés implicada en la transformación de la energía proveniente del combustible en trabajo es el mecanismo biela-manivela. Este mecanismo permite transformar un movimiento alternativo, o lineal, en un movimiento circular o viceversa; gracias a esto podemos aprovechar la expansión producida por la combustión para transformarlo en el movimiento circular que nos permiten mover el eje del motor. Este mecanismo biela-manivela es lo que en un motor se entiende como pistón, formado por el émbolo, elemento cuya superficie superior forma parte de la cámara de combustión y que se une con el otro elemento, la biela, que transmite el movimiento al cigüeñal o eje del motor.

Un motor es mucho más que todo lo que encierra el cilindro, estando formado por un gran conjunto de elementos, sobre todo mecánicos, que juntos conforman esta máquina que nos ha permitido desarrollarnos tanto en los ámbitos industriales y económicos.

Con la actual tendencia de evolución de los motores actuales basada en la mejora de los rendimientos y la reducción en las emisiones de gases contaminantes, se han desarrollado nuevas tecnologías como la tecnología TJI (Turbulent Jet Ignition o Encendido por Chorro Turbulento) o la EGR de baja presión. A pesar de que estas tecnologías se están utilizando en pocos y exclusivos motores sirven de antecedentes para visualizar un futuro en el que esta tecnología alcance un punto de funcionamiento de tal forma que se pueda introducir al mercado, lo que implicaría una reducción del impacto medioambiental que generan los MCIA a día de hoy. Los motores de estudio de este trabajo no son motores convencionales que podamos encontrar en los vehículos comerciales, esto es porque todavía se encuentran en la fase de desarrollo. La aplicación de modelos basados redes neuronales se va a realizar para dos motores, en concreto un motor gasolina

TJI y otro motor diésel con una válvula de EGR de baja presión; pese que para la aplicación de redes neuronales no es necesario ser conocedor de las físicas del sistema, tan solo son necesarias unas entradas y unas salidas, se va a realizar una breve introducción sobre las tecnologías que caracterizan a estos motores para conocer el problemas que nos incumbe y poder ser críticos con los resultados obtenidos.

Descripción de la tecnología TJI

La tecnología TJI fue desarrollada por la empresa alemana Mahle, empresa dedicada al desarrollo y producción de motores. La tecnología TJI fue utilizada por primera vez en la Fórmula 1, haciendo uso de ella por primera vez la escudería Mercedes y posteriormente por otros equipos como Ferrari o Renault.

El motor TJI es un motor monocilíndrico de cuatro tiempos con tecnología pre-cámara y que tiene las siguientes características:

Volumen desplazado	404 cc
Carrera	80.5 mm
Diámetro del pistón	80 mm
Ratio de compresión	13.6:1
Tipo de combustión	SI-TJI
Número de cilindros	1
Número de válvulas/cilindro	2 admisión y 2 escape
Sistema de inyección de combustible	PFI P_{max} 6 bar

Tabla 1.1: Características de motor TJI

La tecnología TJI nos permite trabajar en motores de gasolina, con mezclas pobres, teniendo un exceso de aire, lo que implica una mayor reducción de emisiones y un aumento de la eficiencia térmica. A grandes rasgos, la tecnología TJI consiste en una pre-cámara conecta con la cámara de combustión en la que se encuentra una mezcla de aire y gasolina en dosado estequiométrico, que al iniciarse la combustión mediante una bujía consigue iniciar la mezcla pobre que se encuentra en la cámara de combustión, acelerando considerablemente el proceso de combustión y por tanto obteniendo mayor energía del proceso de combustión. Esto es un resumen muy breve de una tecnología muy prometedora, se puede encontrar más información en Toulson y col., 2010.

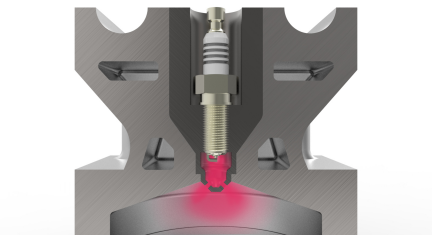


Figura 1.2: Representación de la tecnología TJI. Fuente: soymotor.com

Descripción de la tecnología EGR de baja presión

La tecnología EGR (Exhaust Gas Recirculation) consiste en una válvula situada entre el colector de admisión y el colector de escape que recircula los gases desde el escape hasta la admisión. Esto implica una reducción de la temperatura de combustión, puesto que al introducir menos oxígeno la combustión tendrá menos energía. La ventaja de reducir la temperatura de la combustión es que se reducen los gases contaminantes, sobre todo los NO_X, pero el echo de recircular gases quemados de nuevo al cilindro implica la entrada de carbonilla procedente de la combustión al cilindro, pudiendo rallar las paredes de este y empeorando notablemente su funcionamiento.

De esta desventaja surge la tecnología EGR de baja presión, donde la válvula se encuentra también entre el colector de admisión y la admisión de escape, con la principal diferencia de que esta vez se toman los gases de escape una vez han pasado por el filtro de partículas y por el radiador de la EGR, donde se enfrían los gases.

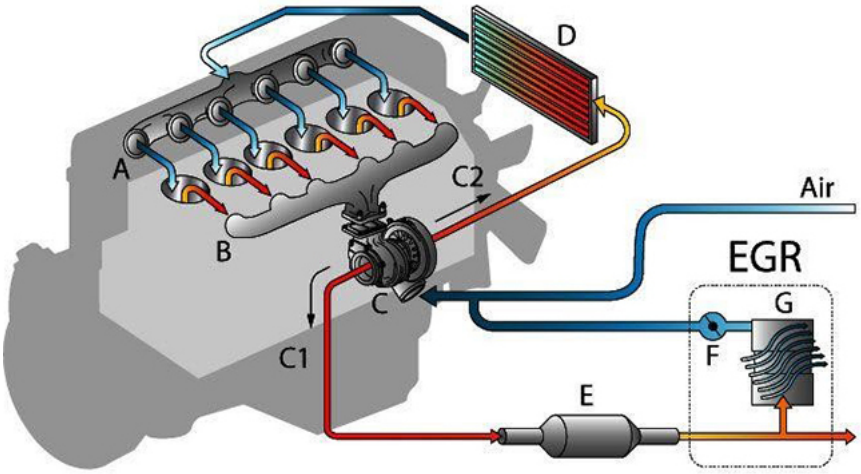


Figura 1.3: Esquema de uso de EGR de baja presión en un motor con turbocompresor. Fuente: autofacil.es

El motor del que se han extraído los datos es un motor EGR de baja presión con las siguientes características:

Volumen desplazado	1968.44 cc
Carrera	47.7 mm
Diámetro del pistón	81 mm
Ratio de compresión	16.2:1
Tipo de combustión	CI
Número de cilindros	4
Número de válvulas/cilindro	2 admisión y 2 escape
Sistema de inyección de combustible	Inyección directa

Tabla 1.2: Características de motor diésel con EGR de baja presión

1.1.2 Introducción Machine Learning

A lo largo de los último años, el Machine Learning y la Inteligencia Artificial han sufrido un gran desarrollo, debido al aumento de la capacidad de cálculo y al desarrollo de mejores algoritmos.

El Machine Learning es, según define IBM, *una rama de la inteligencia artificial (IA) centrada en la creación de aplicaciones que aprenden de los datos y mejoran su precisión con el tiempo sin estar programadas para ello.*

El origen de la inteligencia artificial se produjo en los inicios de los años cincuenta, con la creación, por parte de Alan Turing, del "Test de Turing". Un test que permite determinar si una máquina era inteligente o tan solo estaba engañando al ser humano, haciéndose pasar por inteligente. Posteriormente, en el año 1952, Arthur Samuel, un ingeniero eléctrico, desarrolló lo que es conocido como el primer programa capaz de aprender. Este programa era capaz de aprender a jugar a la damas, llegando a vencer a los campeones de damas en la década de 1960. Más adelante, en el año 1956, se organizó *A proposal for the Dartmouth Summer Research Project on Artificial Intelligence*, 1955, una conferencia en la que se acuñó por primera vez el nombre de Inteligencia Artificial. Gracias a la estadística, el Machine Learning adquirió fama en la década de 1990. Debido a la unión de estos dos campos junto a las grandes bases de datos disponibles, los científicos comenzaron a desarrollar nuevos modelos cada vez más precisos y con mayor capacidad, capaces de resolver problemas más complejos y que se adaptaban mejor a la realidad.

Dentro del Machine Learning existen tres tipos principales: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje reforzado. La principal distinción entre ellos es el modelo y el tipo de entrenamiento, siendo capaces de solucionar distintos problemas.

El aprendizaje supervisado consiste en el entrenamiento del modelo en base a la solución, es decir, se conoce la salida del modelo en función de nuestras entradas, de tal forma que durante el entrenamiento el modelo se ajustará para que, a partir de las entradas del modelo, obtengamos las salidas que conocemos.

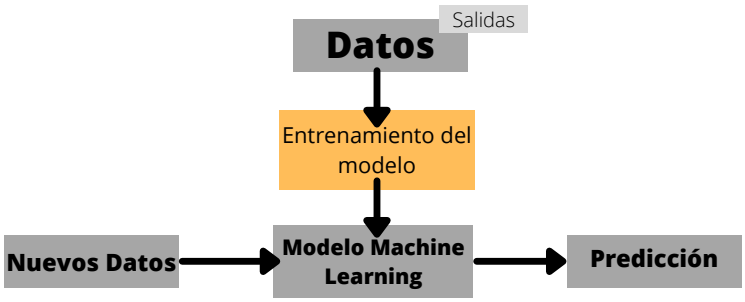


Figura 1.4: Funcionamiento de modelo supervisado. Fuente: elaboración propia

El aprendizaje supervisado soluciona, principalmente, dos problemas. Por una parte el problema de clasificación, en el que debemos ser capaz de, para una entrada, definir de que tipo va a ser la salida. Este problema es muy común en el campo de la visión artificial, pudiendo resolver problemas desde la clasificación de vehículos hasta la detección de enfermedades en los bosques, basándose solo en la información proporcionada por los píxeles de una imagen.

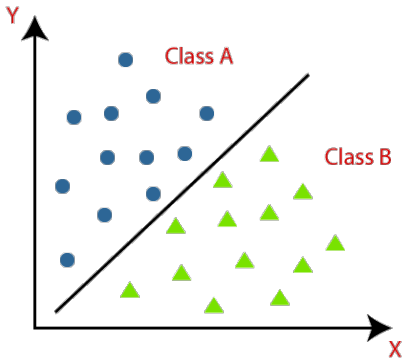


Figura 1.5: Ejemplo de clasificación. Fuente: paperswithcode.com

Por otra parte tenemos el problema de la regresión, siendo este problema el que se abarca en este trabajo, donde en base a unas variables tenemos una salida, esta salida debe ser continua, de tal forma que nuestro modelo se comportará como una función compleja que en base a una entrada nos da una salida.

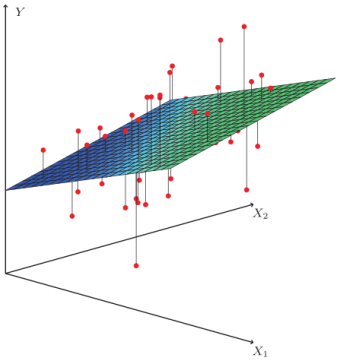


Figura 1.6: Ejemplo de regresión. Fuente: Hastie y col., 2001

El siguiente tipo de Machine Learning es el aprendizaje no supervisado, en este aprendizaje se tienen unas entradas pero no las salidas, de tal forma que será el propio modelo el que determine estas, extrayendo información significativa de nuestros datos de entrada. Mediante este tipo de entrenamiento se pueden resolver el problema de clustering, este problema consiste en organizar grupos de información sin ser conocedores de eso. Un ejemplo de clustering podría ser clasificar distintos tipos de viviendas en función de distintos parámetros como podrían ser el número de habitaciones, precio, metros cuadrados...

El otro problema que se resuelve mediante el aprendizaje no supervisado es la reducción de dimensiones de un gran número de datos. Tener un modelo, con un gran número de entradas, lo dota de una gran complejidad y por tanto de la necesidad de una mayor capacidad de cálculo para su entrenamiento. Reducir el número de entradas, eliminando aquellas que sean, por ejemplo, dependientes de otras, simplifica el modelo, eliminando el ruido de los datos.

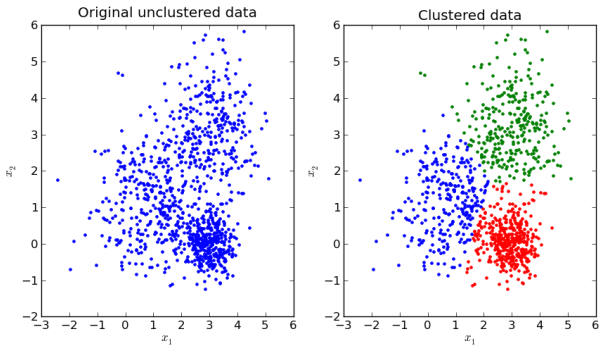


Figura 1.7: Ejemplo de clustering. Fuente: statdeveloper.com

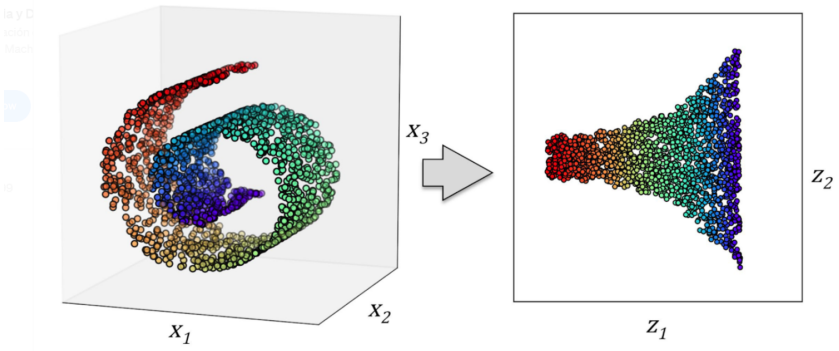


Figura 1.8: Ejemplo de reducción dimensional. Fuente: medium.com

Finalmente, el último tipo de Machine Learning es el entrenamiento reforzado, el cual consiste en situar nuestro modelo en un entorno que lo retro-alimente, de tal forma que el modelo aprenda en base a las respuestas que le da el entorno. Esto es muy utilizado en el ámbito de los juegos, dónde existen distintos modelos capaces de ganar a los mejores jugadores del mundo en juegos como el ajedrez o el Go.

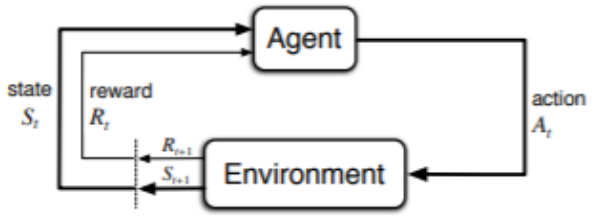


Figura 1.9: Estructura de entrenamiento reforzado. Fuente: Sutton y Barto, 2014

1.2 Objetivos

La cantidad de proyectos para la investigación y desarrollo de nuevas tecnologías en el campo de la automoción es innumerable. Además, dada la relevancia en el mundo actual que tienen estos, cada alternativa debe estudiarse en detalle, con el fin de lograr un uso que no de lugar a fallos y con el que se esté seguro de que implique una mejora del motor.

El proyecto del que forma parte este TFG consiste en la implementación de redes neuronales adaptativas con el fin de optimizar el funcionamiento de los motores de combustión interna alternativa que se encuentran en desarrollo actualmente. Estos motores, como pueden ser un motor TJI (*Turbulent Jet Injection*) o un motor con EGR de baja presión, dónde la expansión de los gases se produce con poco gradiente de presión, son muy difíciles de predecir mediante modelos físicos, es por eso que se plantea el uso de nuevas tecnologías que nos permitan modelar correctamente su funcionamiento, de ahí la creciente necesidad del uso de una tecnología, que ha tenido un enorme desarrollo en los últimos años, como es la inteligencia artificial.

Con el fin de aplicar esta tecnología en estos nuevos motores, el TFG se centra en la predicción de una serie de parámetros que no son fáciles de medir durante el funcionamiento normal de un motor, y que deben ser medidos en un banco de ensayos, utilizando aquellos datos que si que podemos obtener fácilmente durante el uso del motor. De esta forma se logra tener al alcance datos que hasta el momento no se tenían durante el funcionamiento con el fin de poder optimizar el uso de este.

Además, se va a estudiar también la adaptación de esta tecnología, de tal forma que sea capaz de entrenarse y realizando mejores predicciones conforme el motor se está utilizando, no quedándose obsoleta y pudiendo adaptarse a todos los cambios que le ocurran al motor.

De manera resumida, los objetivos del trabajo son:

- El desarrollo y estudio de distintos algoritmos basados en redes neuronales, estudiando el tamaño de las redes, los tiempos de ejecución, su robustez a errores en la medidas y el resto de parámetros que las definen.
- La adaptación de estas redes neuronales a nuevos ciclos mediante distintos tipos de entrenamiento.

1.3 Metodología

La predicción mediante el uso de modelos basados en redes neuronales artificiales requiere del uso de datos precisos que se adecuen a la realidad. Para ello, se han realizado ensayos sobres distintos motores, de tal forma que se obtienen datos de calidad para el entrenamiento y validación de nuestros modelos.

1.3.1 Obtención de datos

La obtención de datos y medidas ha sido realizada por el personal del Instituto Universitario de Motores Térmicos (CMT) de la Universidad Politécnica de València. Los datos provienen de distintos motores que están disponibles en el CMT, y sobre los cuales se han realizado una serie de ensayos, como puede ser un un ciclo NEDC, ciclo que se realiza para evaluar objetivamente el impacto medio-ambiental de los vehículos. Esta toma de datos no forma parte de las tareas relacionadas con el TFG, si no que la ocupación de este trabajo comienza con los datos en bruto o tras algún leve post-procesado.

1.3.2 Modelado, entrenamiento y validación

La totalidad del modelado y la implementación se ha realizado mediante el software de programación orientado a las matemáticas MATLAB. Para la preparación del uso de los datos, en algunos casos ha sido necesario la realización de algunos cálculos, que se comentarán más adelante. Una vez se han tenido los datos necesarios se ha procedido a realizar una limpieza de estos, obteniendo los datos ya limpios que puedan utilizarse para el entrenamiento del modelo. Tras entrenarse el modelo el siguiente paso ha sido la validación de estos, para ver su comportamiento, y finalmente se ha procedido a realizar un estudio de como estas redes consiguen adaptarse ante nuevos datos.

Capítulo 2

Modelado de redes neuronales adaptativas

Este capítulo está destinado a establecer el marco teórico en el que vamos a trabajar, dejando definido todo lo referente al modelado de las redes neuronales, incluyendo desde su definición a como van a ser entrenadas. De tal forma que de cara a la implementación de éstas quede todo correctamente definido.

Tal y como se ha comentado en el apartado 1.1.2, una aplicación del Machine Learning era la regresión, esto nos permite, en base a unas entradas en nuestra red ser capaz de determinar una salida que se ajuste a la realidad. Para ello se va a hacer uso de las redes neuronales artificiales, una idea basada en el funcionamiento del cerebro de los organismos vivos, dónde una serie de neuronas se unen y se va transmitiendo la información entre ellas. Con el paso del tiempo estas neuronas y sus conexiones van adaptándose para conseguir aprender.

2.1 Neuronas

Uno de los principales elementos de las redes neuronales son las neuronas, en los organismos vivos, como se aprecia en la figura 2.1, las neuronas están formadas por tres partes, las dendritas, encargadas de recibir los estímulos externos o procedentes de otras neuronas, el soma, que se encarga de procesar estos estímulos y el axón, que se encarga de transmitirlos.

Basándose en las neuronas de los seres vivos, en 1943, McCulloch y Pitts, como se explica en McCulloch y Pitts, 1943, desarrollaron el primer modelo neuronal, la Neurona de McCulloch-Pitts, esta neurona hacía la suma ponderada de las entradas con una posterior función de activación, que permitía realizar funciones lógicas.

A la neurona le entran una serie de valores $x_1, x_2, ..., x_n$, que son multiplicados por los pesos que se le asignaban a cada entrada w_1, w_2, w_n, siendo el número de pesos igual al número de entradas, además se le añade una un valor umbral θ que controla el sesgo de la neurona. A continuación se multiplican las entradas por los pesos:

$$w_1x_1 + w_2x_2 + ... + w_nx_n - \theta = w^T x \tag{2.1}$$

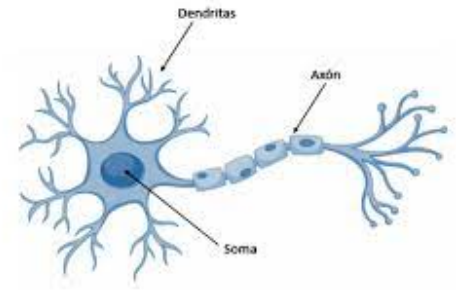


Figura 2.1: Representación de una neurona de un ser vivo. Fuente: xeridia.com

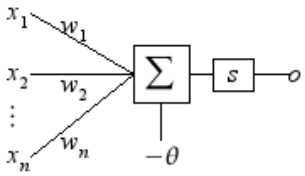


Figura 2.2: Diagrama de una Neurona de McCulloch-Pitts. Fuente: wikipedia.org

siendo:

$$w = (w_1, w_2, ..., w_n, -\theta)^T \quad \text{y} \quad x = (x_1, x_2, ..., x_n, 1)^T \tag{2.2}$$

Posteriormente la salida de esta multiplicación se pasa por la función de activación s, que para la Neurona de McCulloch-Pitts, es la siguiente:

$$s(u) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.3}$$

de tal forma que la salida de la neurona sería la siguiente:

$$y = s(w^T x) \tag{2.4}$$

El modelo desarrollado por McCullich y Pitts se llevó más allá por parte de Frank Rosenblatt, desarrollando el perceptrón. El perceptrón toma las bases de la Neurona de McCullich y Pitts y desarrolla una modelo cuya función era la de clasificar patrones linealmente separables, una aplicación muy clara y utilizada de este modelo es la representación de funciones lógicas variando los pesos.

Como se puede ver en la figura 2.3 la estructura del perceptrón es exactamente igual a la de la Neurona de McCulloch-Pitts (figura 2.2), la principal diferencia es que el desarrollo del perceptrón fue pensado para entrenarse, de tal forma que fuera capaz de realizar cualquier clasificación lineal, de este entrenamiento se hablará más adelante en el apartado 2.3.3. Otra gran diferencia es que el perceptrón tan solo puede tener entradas booleanas, es por esto que es de gran aplicación para representar funciones lógicas.

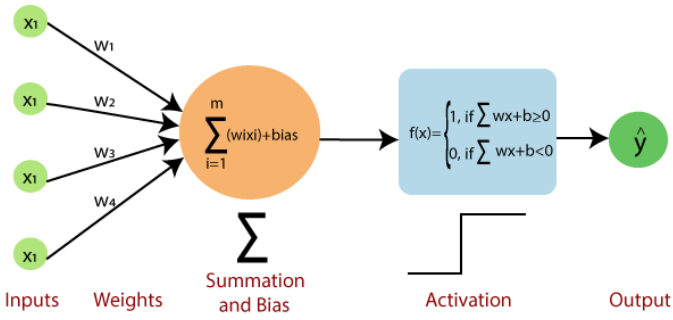


Figura 2.3: Diagrama de un perceptrón. Fuente: javatpoint.com

2.2 Función de activación

Hasta el momento se ha hablado de la función de activación como una función, cuya ecuación es 2.3, que en función de si la entrada era mayor o menor de cero se obtenía una salida booleana. Esta función de activación es una función de umbral, o función escalón; pero existen diversas funciones que nos permiten dejar atrás la condición de que nuestros datos, o bien a clasificar, o bien a ajustar, sean lineales; pudiendo obtener un modelo más complejo que se adapte mejor a ellos.

A lo largo de este apartado se tratarán las principales funciones de activación. De estas funciones de activación cabe destacar tres características, la primera es su no linealidad, el porqué esto es un problema se tratará más adelante en el desarrollo de la función lineal. La segunda característica es el rango en el que trabaja la función, ya que cuando la función tiene un rango finito los algoritmos de entrenamiento basados en el descenso del gradiente se comportan de una forma más robusta. Finalmente, la tercera característica es la diferenciabilidad de la función, es de gran interés que sea diferenciable en todo el rango de la función, para poder aplicar algoritmos basados en el descenso del gradiente. Todo esto cobrará más sentido en el apartado de entrenamiento 2.3.3, donde se detalla el uso de algoritmos basados en el descenso del gradiente.

Dentro de las funciones de activación existen distintos tipos. Primero trataremos la función lineal, o identidad, que sigue la siguiente ecuación:

$$f(x) = x \tag{2.5}$$

y cuya representación gráfica es la siguiente:

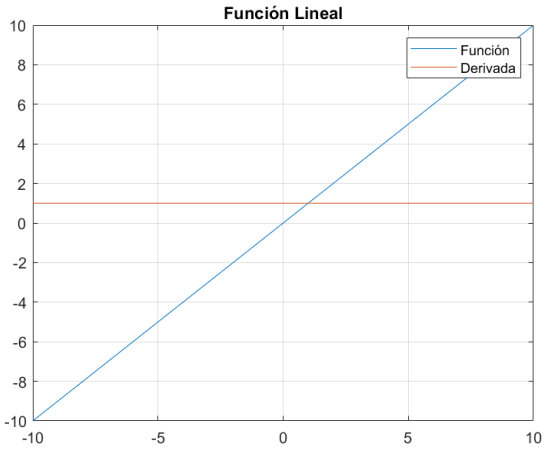


Figura 2.4: Función lineal y derivada de la función lineal. Fuente: elaboración propia

El principal problema de estas función es que al ser lineal es incapaz de funcionar como un clasificador universal. Un clasificador universal es aquel que es capaz de clasificar los datos independientemente de si existe, o no, una correlación lineal entre ellos, esto fue demostrado por Hornik y col., 1989.

A continuación tenemos la *Rectified Linear Unit* o ReLU, esta función es muy similar a la función lineal, pero con un pequeña diferencia, y es que si la entrada a la función es ≤ 0 la función toma el valor 0, siendo su expresión matemática la siguiente:

$$f(x) = max(0, x) \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ \text{undefined} & \text{if } x = 0 \end{cases} \tag{2.6}$$

y cuya representación gráfica es la siguiente:

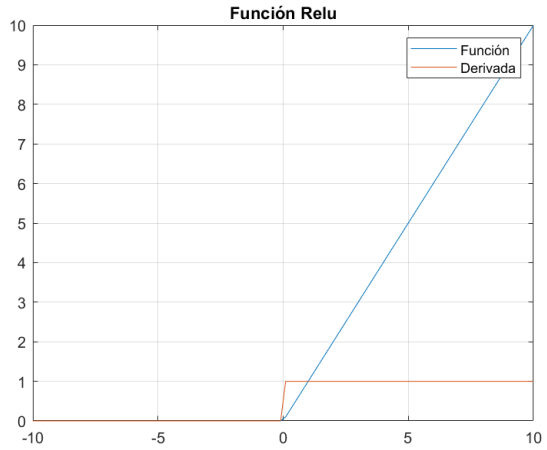


Figura 2.5: Función ReLU y derivada de la función ReLU. Fuente: elaboración propia

A diferencia de la función lineal, esta función si que funciona como clasificador universal, pero el mayor problema de esta función es que no es diferenciable en todo su rango, es fácil ver que cuando $x = 0$ no podemos calcular su derivada, de tal forma que pese a poder aplicarse algoritmos basados en el descenso del gradiente, podrían surgir problemas en ese punto.

Debido a la existencia de este problema surge una nueva función de activación, que suaviza la función y la convierte en diferenciable en todo su dominio. Esta es la función *Softplus*, cuya ecuación es:

$$f(x) = \ln(1 + e^x) \tag{2.7}$$

Esta función si que es diferenciable en todo su dominio, siendo su derivada:

$$f'(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

y la representación gráfica, tanto de la función como de la derivada:

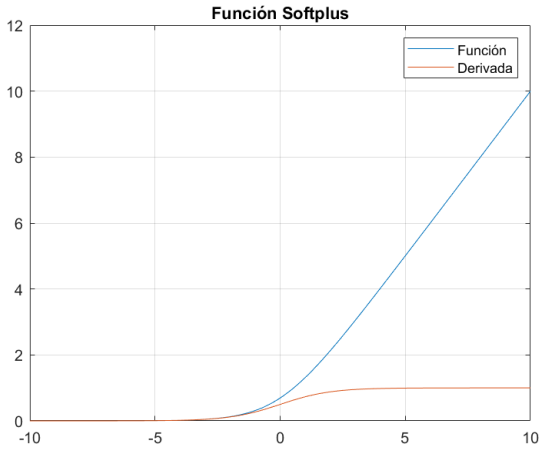


Figura 2.6: Función Softplus y derivada de la función Softplus. Fuente: elaboración propia

Como se ha comentado al inicio de este apartado, es de gran interés para el algoritmo del descenso del gradiente el uso de funciones de activación que tengan un rango acotado, al ser el rango de esta función de activación $(0, \infty)$, esto hará que los algoritmos de entrenamientos basados en el descenso del gradiente no se comporten de la manera más adecuada.

La siguiente función de activación a estudiar va a ser la función logística, o más conocida como función *sigmoide*, y que sigue la siguiente ecuación:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.9}$$

siendo su derivada:

$$f'(x) = f(x)(1 - f(x)) \tag{2.10}$$

y la representación gráfica, tanto de la función como de la derivada:

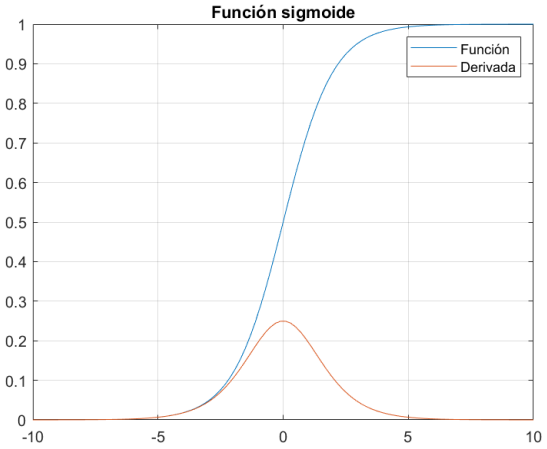


Figura 2.7: Función sigmoide y derivada de la función sigmoide. Fuente: elaboración propia

Esta función de activación es la más sencilla de implementar, dado que su derivada se puede simplificar mucho, además es diferenciable en todo su dominio, y su rango está acotado entre $(0, 1)$. Esto hace que sea una opción muy interesante a tener en cuenta para implementar en nuestra red neuronal.

De la función sigmoide se deriva la función SoftMax, en la cual no vamos a definir con profundidad, pues es de mayor complejidad, y la mayor diferencia respecto a la sigmoide es que la suma de la salida de todas las neuronas es igual a uno. Esto lo hace muy interesante para problemas de clasificación, ya que otorga una probabilidad a cada una de nuestras salidas.

Por último, otra función de gran interés es la de la tangente hiperbólica, cuya formula es:

$$f(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.11}$$

su derivada:

$$f'(x) = 1 - f(x)^2 \tag{2.12}$$

y la representación gráfica, tanto de la función como de la derivada:

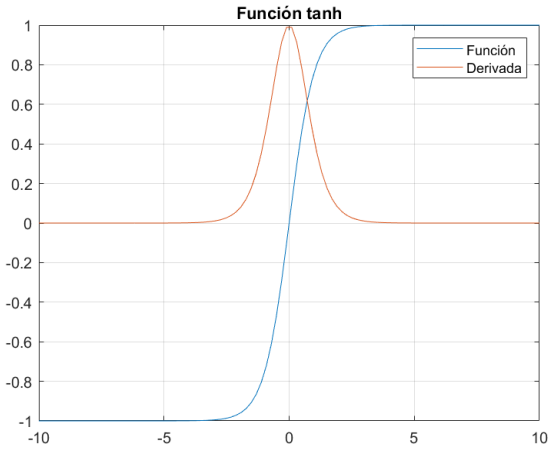


Figura 2.8: Función tangente hiperbólica y derivada de la tangente hiperbólica. Fuente: elaboración propia

Esta función es de gran interés porque cumple todos los requisitos nombrados en el inicio de este apartado, es una función diferenciable en todo su dominio y que tiene un rango acotado entre $(-1, 1)$.

2.3 Redes Neuronales

Una vez ya conocemos el principio básico de las neuronas y la función de activación que vamos a utilizar se va a empezar a detallar la estructura de la red, con todos los parámetros que la conciernen, a continuación se estudiará como, a partir de la estructura de la red definida, se obtiene una salida mediante un algoritmo llamado Forward Propagation, y finalmente el entrenamiento de estas redes.

2.3.1 Estructura de la red

La estructura de la red es algo que va a ser muy característico de nuestro modelo. Una red neuronal está formado por una serie de neuronas conectadas entre si, estas neuronas se clasifican en distintas capas, y a su vez éstas se separan en la capa de entrada, capas ocultas y capa de salida.

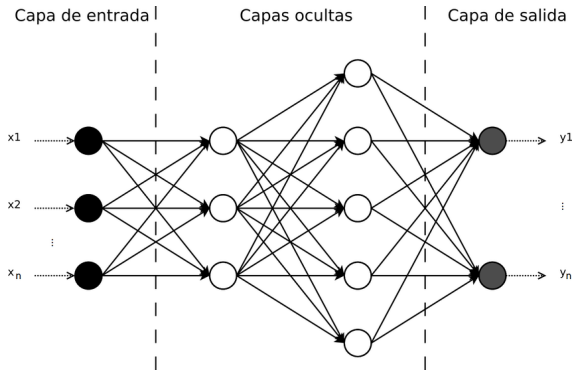


Figura 2.9: Diagrama de una Red Neuronal Artificial. Fuente: researchgate.net

La capa de entrada será por la cual los valores de entrada se introducirán en el modelo de la red neuronal, de tal forma que tengamos tantas neuronas en la capa de entrada como entradas tenga nuestra red, o mejor dicho el número de características de nuestros datos:

$$x = (x_1, x_2, ..., x_n) \tag{2.13}$$

siendo n el número de características de los datos. Estos datos se van desplazando hacía delante hasta llegar a la capa de salida, esto lo estudiaremos más adelante en el apartado 2.3.2.

Estos datos es ideal que estén normalizados entre $(0, 1)$, para que ninguna característica tenga un mayor peso que otras.

Tanto el número de capas ocultas de nuestra red como el tamaño de ésta se determinan de manera experimental, teniendo en cuenta que cuanto mayor sean éstas, más compleja será la red. Esto hará que se adapte mejor a nuestros datos pero implicará un mayor coste computacional y por tanto un mayor tiempo de entrenamiento, debiendo decidir si se está dispuesto a sacrificar algo de precisión por un mayor coste temporal, esto se estudiará más adelante en el capítulo 3.

Independientemente de esto, hay que aclarar un par de conceptos en cuanto a la estructura de la red. Para ello es necesario explicar los conceptos de overfitting y underfitting. Como se verá más adelante, cuando se hable del entrenamiento en el apartado 2.3.3, se realizan una serie de iteraciones sobre nuestro modelo para reducir el error del modelo, para ello se usan los datos que queremos ajustar, estos datos se separan en un gran bloque para el entrenamiento y otro para la validación, gastando los datos de entrenamiento para esto. El overfitting se denomina al problema en el que nuestro modelo se ajusta de manera excesiva a nuestros datos de entrenamiento, obteniendo un modelo con un error muy pequeño en los datos de entrenamiento pero que no se ajusta a la realidad, de tal forma que el error en los datos de validación es mucho mayor al de los datos de entrenamiento. Lo contrario ocurre con el underfitting, en este caso, un ajuste muy débil a los datos de entrenamiento nos lleva a un modelo en el que nuestro modelo no se ajusta correcto a la realidad, obteniendo un gran error tanto con los datos de entrenamiento como con los de validación.

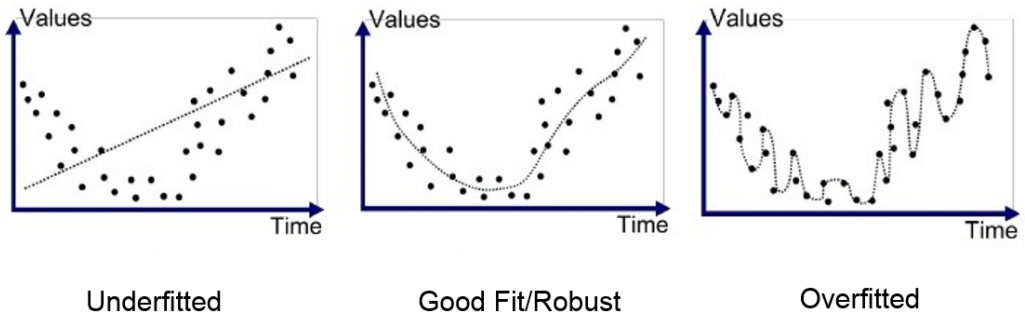


Figura 2.10: Ejemplos de Overfitting, Underfitting y un correcto entrenamiento. Fuente: medium.com

El uso de un modelo de redes neuronales demasiado complejo puede llevar a problemas de overfitting, ya que cuanto más parámetros tenga nuestra red, mayor será la capacidad de ajuste de nuestros datos al modelo, pero este sobre ajuste puede alejarlo de la realidad. Con el uso de un

modelo de red neuronal demasiado sencillo ocurre lo contrario, el no tener la capacidad de ajuste a los datos puede acarrear problemas de underfitting.

Para finalizar con la definición de la estructura de la red, nos queda hablar de la conexión de las neuronas. Las neuronas de distintas capas están conectadas entre sí por los parámetros de la red, o más conocidos como pesos. Estos pesos serán los que ajustaremos durante el entrenamiento para reducir el error de nuestro modelo. Los pesos que conectan cada capa de la red formarán una matriz cuyo tamaño dependerá de las neuronas de las capas que conecte, de tal forma que

$$W = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,j} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ w_{i,1} & w_{i,2} & \cdots & w_{i,j} \end{pmatrix} \tag{2.14}$$

siendo i el número de neuronas de la capa anterior y j el número de neuronas de la capa siguiente.

2.3.2 Forward Propagation

Con la definición de toda la estructura de la red es momento del estudio de como en base a unos datos de entrada la red calcula una salida para ellos. Esto se realiza mediante el algoritmo de Forward Propagation, o propagación hacia delante.

Para aplicar este algoritmo es necesario haber definido toda la estructura de la red, por ello, partimos de la base de que se ha definido lo siguiente:

- Unos datos de entrada $X = (x_1, x_2, ..., x_n)^T$, siendo n el número de características de estos datos de entrada.
- Un número de capas m, de tal forma que la primera capa sea la capa de entrada y la capa m se la capa de salida. Esta primera capa tendrá n neuronas y la capa de salida tendrá p neuronas, siendo p el número de parámetros que queremos predecir.
- Unas matrices de pesos W, en concreto $m - 1$ matrices, que conectaran las neuronas de distintas capas entre sí.
- Una función de activación $g(z)$, que se aplicará en todas las neuronas de nuestra red.

Con todo esto definido vamos a desarrollar el algoritmo de Forward Propagation:

Para entender mejor el algoritmo se va a desgranar un poco. En primer lugar se tienen que los datos de entrada (X) se convierten en las salidas de las neuronas de la capa de entrada,

$$out_1 = X \tag{2.15}$$

para avanzar hacia las siguientes neuronas debemos multiplicar por los pesos que conectan la capa de entrada con la primera capa oculta (W_1).

$$input_2 = out_1 \cdot W_1 \tag{2.16}$$

Algorithm 1 Forward Propagation

Require: Datos de entrada $X = (x_1, x_2, ..., x_n)^T$, $m-1$ matrices de pesos W, función de activación $g(z)$.

Ensure: Predicción del modelo $Y_{modelo} = (y_1, y_2, ..., y_p)^T$.

```
input = X^T
for i = 1 hasta m − 1 do
    if i = m − 1 then
        Y_modelo = g(input · W_i)
    else
        input = g(input · W_i)
    end if
end for
return Y_modelo
```

Esto sería la entrada a las neuronas de la primera capa, y para calcular sus salidas debemos aplicarle su función de activación ($g(z)$), de tal forma que la salida sería:

$$out_2 = g(input_2) \tag{2.17}$$

esta salida sería la entrada de la siguiente capa, y los pasos para calcular su salida serían los mismo, de tal forma que se iría avanzando hasta llegar a la última capa, en la que la salida de estas sería la predicción del modelo,

$$Y_{modelo} = out_m = g(out_{m-1} \cdot W_{m-1}) \tag{2.18}$$

con esto ya tendríamos la salida de la red neuronal.

2.3.3 Entrenamiento

El proceso de entrenamiento de una red neuronal consiste en hacer variar los pesos (W), de tal manera que el ajuste de estos implique una salida de nuestro modelo con menor error. El error del modelo de la red neuronal es lo que trataremos de minimizar, este error es entre la salida de nuestro modelo y la salida conocida, para unos datos de entrada dados, de tal forma que la función del error viene dada por:

$$E = \frac{1}{2} \sum_{k=1}^{p} (y_{modelo} - y)^2 \tag{2.19}$$

siendo p el número de características a predecir, y_{modelo} la salida predicha por el modelo y y la salida conocida. Es necesario destacar que todos los datos de salida Y deben estar normalizados, esto depende de la función de activación que vayamos a utilizar y su rango, se detallará más adelante en el apartado referente a la implementación (3).

De está función de error puede destacar la presencia de la constante $\frac{1}{2}$ multiplicando al sumatorio, esto es debido a que durante el entrenamiento se calculará la derivada del error, por tanto esta constante después se cancelará, quedando algo más simplificada.

Para minimizar este error existen distintos algoritmos, solo vamos a estudiar el algoritmo conocido como descenso del gradiente. Esto es debido a que este algoritmo es comúnmente utilizado en el ámbito de la inteligencia artificial por los pocos recursos que consume y su facilidad de implementación. Si bien es cierto que no es el algoritmo más rápido ni el que nos asegura que el error alcance el mínimo global será este el que utilizaremos.

Backpropagation

Antes de presentar el algoritmo de Backpropagation, se debe tener claro que con el algoritmo de entrenamiento lo que se busca es variar los pesos (W) para minimizar la función de coste, en nuestro caso el error (Ecuación 2.19). Para ello lo que se hace es calcular para cada peso la derivada del error respecto a este, de tal forma que se conoce en que medida afecta ese valor del peso al error total, variando en mayor medida aquellos cuya derivada sea mayor. De tal forma que el algoritmo de Backpropagation es una forma de calcular estas derivadas parciales.

Antes de explicar la implementación de este algoritmo se va a hacer una pequeña explicación teórica sobre él. Para hacerlo nos apoyaremos en una red neuronal muy sencilla, en concreto una red neuronal con una capa de entrada con dos neuronas, una capa oculta con otras dos neuronas y una capa de salida de dos neuronas. Además se introduce el concepto de bias, esto es una forma de añadir variabilidad al modelo, añadiendo una neurona que solo está conectada con las neuronas de delante y cuyo valor será siempre uno, y cuyos pesos una vez se inicializan no varían. También se debe definir una función de activación para sus neuronas, en este ejemplo se gasta la función sigmoide, cuya función es 2.9. De tal forma que la estructura de la red que obtenemos es la siguiente:

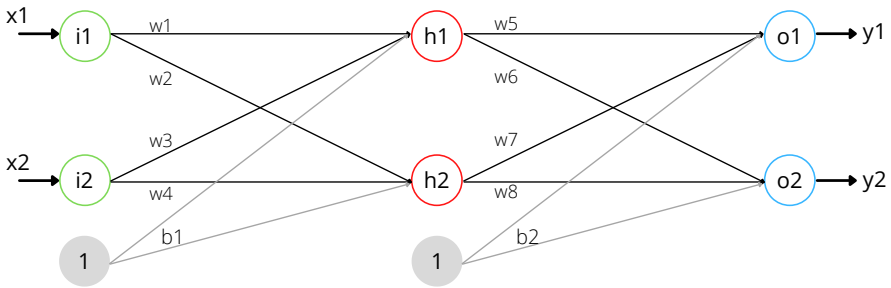


Figura 2.11: Ejemplo de red neuronal Fuente: elaboración propia

Lo que se busca calcular son las derivadas del error de la red respecto a cada peso:

$$\frac{\partial E}{\partial w_i} \tag{2.20}$$

siendo el error el definido anteriormente 2.19. Para calcular estas derivadas, se comienza por las más cercanas a la capa de salida, por ejemplo w_5, y aplicamos la regla de la cadena:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out_{O_1}} \frac{\partial out_{O_1}}{\partial input_{O_1}} \frac{\partial input_{O_1}}{\partial w_5} \tag{2.21}$$

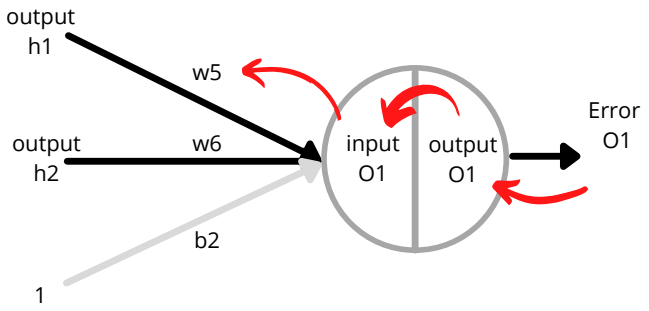


Figura 2.12: Ejemplo de Backpropagation Fuente: elaboración propia

Ahora ahondaremos en estas derivadas. En primer lugar, el error E, particularizando para el caso de estudio será:

$$E = \frac{1}{2}((y_{O_1} - out_{O_1})^2 + (y_{O_2} - out_{O_2})^2) \tag{2.22}$$

si calculamos la derivada parcial del error respecto a la salida de la neurona O_1 obtenemos:

$$\frac{\partial E}{\partial out_{O_1}} = -(y_{O_1} - out_{O_1}) \tag{2.23}$$

Siguiendo con la derivada de la salida de la neurona O_1 respecto a su entrada, y sabiendo que su función de activación es la sigmoide:

$$\frac{\partial out_{O_1}}{\partial input_{O_1}} = \frac{\partial(\frac{1}{1+e^{-input_{O_1}}})}{\partial input_{O_1}} = out_{O_1}(1 - out_{O_1}) \tag{2.24}$$

Tras este cálculo se entiende porque una característica de las funciones de activación es que fuera diferenciable en todo su dominio. A la hora de calcular la derivada del error la derivada de la función de activación se ve implicada, pudiendo surgir problemas en el algoritmo si no fuera diferenciable en todo su dominio. Este problema se puede resolver aplicando un algoritmo conocido como descenso del gradiente estocástico, pero no se entrará en él puesto que será el algoritmo del descenso del gradiente el que se utilice, dada su sencilla interpretación y su escasa potencia de cálculo necesaria.

Siguiendo con las derivadas tan solo queda la derivada de la entrada de la neurona respecto a los pesos, sabiendo que esta entrada en función de los pesos es:

$$out_h^t W_2 = out_{h_1} w_5 + out_{h_2} w_6 + b_2 \tag{2.25}$$

luego su derivada será:

$$\frac{\partial input_{O_1}}{\partial w_5} = out_{h_1} \tag{2.26}$$

de tal forma que tras aplicar la regla de la cadena se obtiene:

$$\frac{\partial E}{\partial w_5} = -(y_{O_1} - out_{O_1}) out_{O_1}(1 - out_{O_1}) out_{h_1} \tag{2.27}$$

si se llama δ_{O_1} a:

$$\delta_{O_1} = \frac{\partial E}{\partial out_{O_1}} \frac{\partial out_{O_1}}{\partial input_{O_1}} = -(y_{O_1} - out_{O_1}) out_{O_1}(1 - out_{O_1}) \tag{2.28}$$

obteniéndose:

$$\frac{\partial E}{\partial w_5} = \delta_{O_1} out_{h_1} \tag{2.29}$$

Siguiendo los mismos pasos para calcular las derivadas parciales del error respecto a los pesos que conectan las capas de entradas con las neuronas anteriores, por ejemplo w_1, se obtiene:

$$\frac{\partial E}{\partial w_1} = (\sum_O \frac{\partial E}{\partial out_O} \frac{\partial out_O}{\partial input_O} \frac{\partial input_0}{\partial out_{h_1}}) \frac{\partial out_{h_1}}{\partial input_{h_1}} \frac{\partial input_{h_1}}{\partial w_1} = (\sum_O \delta_O w_{hO}) out_{h_1}(1 - out_{h_1}) i_1 = \delta_{h_1} i_1 \tag{2.30}$$

Repitiendo esto para el resto de los pesos ya se tendrían todas las derivadas de los pesos calculadas y se podría continuar con el algoritmo de entrenamiento.

Visto este pequeño ejemplo y ya sabiendo en que consiste el algoritmo Backpropagation se va a desarrollar adecuadamente. Para ello, al igual que con el algoritmo Forward Propagation (2.3.2), es necesario definir una seria de variables previas que tomará el algoritmo, estas son las mismas que en el algoritmo Forward Propagation (2.3.2), añadiendo lo siguiente:

- Será necesario conocer el error de la red para los datos dados, este error es el que se ha nombrado anteriormente (2.19).
- También será necesario conocer el valor de la derivada de la salida respecto a la entrada de cada neurona ($\frac{\partial out}{\partial input}$), esto estará formado por $m - 1$ matrices.
- Finalmente, la salida del algoritmo que serán $m - 1$ matrices de δ, donde se almacenarán los valores δ, explicados anteriormente.

Con esto definido pasamos al desarrollo del algoritmo:

Algorithm 2 Backpropagation

Require: Error del modelo para unos datos dados $error$, $m-1$ matrices de pesos W, $m-1$ matrices de $\frac{\partial out}{\partial input}$.

Ensure: Matrices δ, en concreto $m-1$ matrices.

```
for i = m − 1 hasta 1 do
    if i = m − 1 then
        δ_i = error · (∂out/∂input)_i
    else
        δ_i = (∂out/∂input)_i δ_{i+1} W_{i+1}
    end if
end for
return δ
```

Descenso del Gradiente

Una vez ya se conoce el algoritmo de Backpropagation (2), se puede avanzar al algoritmo del descenso del gradiente. Este algoritmo busca minimizar la función de costes variando los pesos, para ello alterará los pesos de la siguiente manera:

$$w^{+} = w - \eta \frac{\partial E}{\partial w} \tag{2.31}$$

siendo w^{+} el nuevo peso actualizado y η un parámetro conocido como learning rate. Mediante este algoritmo se darán pasos en dirección hacía donde el gradiete sea mayor, estos pasos serán de mayor o menor tamaño en función del learning rate.

Estos pesos se irán actualizando, de forma iterativa, hasta que, o bien se cumpla un número de iteraciones que nosotros queramos, o bien se alcance un error mínimo indicado por nosotros. Al número de iteraciones máximas que vamos a permitir ejecutar el algoritmo se le llamará it_{max} y al error mínimo para que terminen las iteraciones err_{min}.

Para comenzar con el algoritmo de entrenamiento primero se deben de definir unos hiper-parámetros, éstos determinarán el comportamiento del algoritmo de entrenamiento y son los siguientes:

- El error mínimo para que terminen las iteraciones err_{min}.
- El learning rate η.
- El número máximo de iteraciones del algoritmo del descenso del gradiente it_{max}.

Una vez se tienen los hiper-parámetros definidos es necesario destacar la inicialización de las matrices de pesos. Estas matrices deben inicializarse dependiendo del tamaño de ésta y de la función de activación, con el fin de no ampliar el objeto de estudio de este trabajo se han seguido las recomendaciones de Kumar, 2017.

Ahora que ya se tiene tanto los hiper-parámetros definidos como todo lo que se ha definido para el algoritmo de Forward Propagation (1) y Backpropagation (2), detallaremos el algoritmo de entrenamiento basado en el descenso del gradiente.

Algorithm 3 Descenso del gradiente

Require: Error mínimo (err_{min}), learning rate (η), iteraciones máximas (it_{max}), vector, de tamaño n, de entrada del modelo (X); vector, de tamaño p, de la salida deseada (Y) y $m-1$ matrices de pesos del modelo (W).

Ensure: Matrices de pesos actualizadas W.

$error = 1$ and $iteracion = 1$
while $error \geq err_{min}$ and $iteracion \leq it_{max}$ **do**
  Forward Propagation
  $error = \frac{1}{2}(Y - Y_{modelo})^2$
  Backpropagation
  for $i = 1$ hasta $i = m - 1$ **do**
    if $i = 1$ **then**
      $W_i = W_i + \eta(X^T \delta_i)$
    else
      $W_i = W_i + \eta(out_i^T \delta_i)$
    end if
  end for
  $iteracion = iteracion + 1$
end while
return W

Para cerrar el apartado del entrenamiento nos falta determinar como entran los datos a nuestra red neuronal durante el entrenamiento. El algoritmo no especifica nada sobre el tamaño de los datos, pero en función de estos podemos tener tres tipos de entrenamiento, conocidos como online learning, batch-learning y mini batch-learning.

El algoritmo de online learning es comúnmente utilizado cuando se tienen un flujo continuo de datos, de tal forma que para cada nuevo dato que le proporciona el entorno, el algoritmo puede gastar este dato para entrenar a la red. El mayor problema de esto es que es un proceso iterativo muy lento, pues para este entrenamiento deberán pasar todos los datos uno a uno, y en el caso de que fueran necesarias más iteraciones para seguir minimizando el error deberían procesarse todos de nuevo.

El siguiente algoritmo, el batch-learning, trata un conjunto de datos de manera matricial, de esta forma, computa todos los datos a la vez, calculando la media del error. Este algoritmo es más rápido de implementar, puesto que el bucle necesario anteriormente, para el online learning, ahora puede transformarse en una operación vectorial. Pero este algoritmo tiene un problema, y es la dirección del gradiente, al calcular la media de los errores, no se tomará la derivada con mayor pendiente y se dejarán de lado datos con mucho error.

Finalmente, se tiene el mini batch-learning, este algoritmo es un híbrido entre los dos anteriores, almacena los datos en pequeñas matrices, comúnmente de 32 elementos, para después procesar cada matriz como si se tratara del batch-learning pero realizando iteraciones como el online learning. De esta forma consigue las ventajas del online learning con una mayor velocidad de entrenamiento.

Durante todo el proceso de entrenamiento gastaremos el mini batch-learning puesto que a lo largo del tiempo se ha tomado como el método de entrenamiento de referencia en la comunidad del machine learning. Por otra parte para el proceso de adaptación se utilizará el online learning.

Habiendo definido los algoritmos necesarios para el entrenamiento de las redes neuronales artificiales, se da por concluido el estudio teórico referente a éstas. En próximos apartados se implementará todo ello para adaptarlo al problema y tratar de encontrarle una solución.

2.4 Extreme Learning Machine

Hasta el momento se ha hablado de redes neuronales artificiales como un proceso iterativo en el que mediante el algoritmo del descenso del gradiente nuestro modelo es capaz de entrenarse poco a poco. Pero en 2005, Huang y col., 2005b desarrollaron lo que hoy en día se conoce como Extreme Leaning Machine (ELM), este nuevo campo está basado en las redes neuronales artificiales, con algunas pequeñas diferencias que hacen que su método de entrenamiento sea mucho más sencillo y por tanto su velocidad de entrenamiento nada comparable a las redes neuronales artificiales. Estas mejoras conllevan un detrimento del ajuste de nuestro modelo a la realidad, aumentando el error del modelo.

2.4.1 Estructura de la red

La principal diferencia de este campo se encuentra en la estructura de la red neuronal. Esta estructura mantendrá la capa de entrada y la capa de salida, pero esta vez solo existirá una única capa oculta. Es decir, la estructura de la red siempre estará formada por una capa de entrada de n neuronas, siendo n el número de características de los datos de entrada, una única capa oculta de m neuronas y una capa de salida de p neuronas que dependerá de las características a predecir. La salida de esta última capa tendrá la función de activación comentada anteriormente, la función lineal (2.5).

La otra gran peculiaridad es que la matriz de pesos que conecta la capa de entrada con la capa oculta (W_1) es inicializada de forma aleatoria entre $(-0{,}5, 0{,}5)$. A diferencia de las redes neuronales artificiales, dónde la inicialización de las matrices de pesos era dependiente de la función de activación que se iba a utilizar, en el ELM siempre inicializaremos la matriz W_1 entre estos pesos.

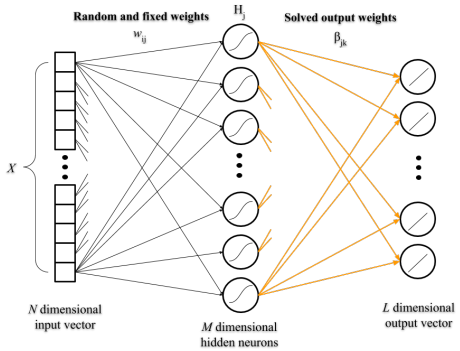


Figura 2.13: Esquema de ELM. Fuente: imcorp.jp

De esta forma la estructura de la red se simplifica considerablemente, además es fácil demostrar empíricamente que a partir de un cierto número de neuronas en la capa oculta no hay una reducción del error del modelo, como se puede observar en la figura 2.14.

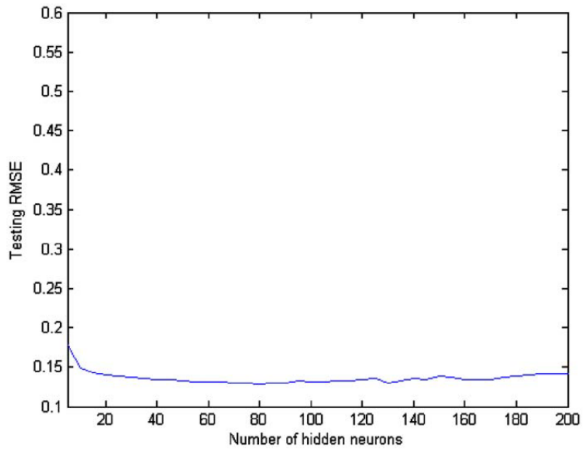


Figura 2.14: Raíz del error cuadrático medio para la predicción de un dataset sobre alojamiento de California. Fuente: Huang y col., 2005b

2.4.2 Forward Propagation

El algoritmo de Forward Propagation es prácticamente igual que para las redes neuronales (1), con la peculiaridad de que la capa de salida tiene una función de activación lineal. De esta forma, para una entrada dada, y las dos matrices de pesos W_1 y W_2, que conectan la capa de entrada con la capa oculta y, ésta última, con la capa de salida, respectivamente; y una función de activación para la capa oculta $g(z)$, usualmente usada la función sigmoide (2.9), la salida del modelo se podrá calcular como:

$$Y = g(W_1 X)W_2 \tag{2.32}$$

Como se puede observar en la ecuación 2.32, calcular la salida de nuestro modelo dados unos datos X es extremadamente rápido de calcular para cualquier ordenador de hoy en día, al haber solo dos operaciones matriciales y el uso de la función de activación.

2.4.3 Entrenamiento

El entrenamiento del ELM es lo más interesante y lo que lo considera una opción interesante en nuestro estudio, y es que, como en todo modelo basado en redes neuronales, el principal objetivo es reducir el error de éste. Para ello, el ELM busca aproximar las salidas esperadas a las salidas predichas de la siguiente forma:

$$min\|HW_2 - Y\| \tag{2.33}$$

siendo $H = g(W_1 X)$.

De esta forma esto se reduce a un problema de mínimos cuadrados, y como demostraron Huang y col., 2005b, esto se puede minimizar, calculando W_2 de la siguiente forma:

$$W_2 = H^+ \tag{2.34}$$

siendo H^+ la generalizada inversa de Moore-Penrose de H, que cumple que:

$$HH^+H = H \tag{2.35}$$

Con la ecuación 2.34 definida, ya se tiene toda la parte del entrenamiento aclarada.

También cabe destacar que, así como en el entrenamiento para las redes neuronales existía el batch learning, online learning y mini-batch learning; el origen del ELM fue planteado únicamente como batch learning, pero más adelante Huang y col., 2005a plantearon el online extreme learning machine (OSELM).

Capítulo 3

Implementación de redes neuronales

A lo largo de este capítulo se va a desarrollar todo lo referente a la implementación de las redes neuronales, justificando el uso de todos los elementos con el fin de obtener un modelo que se aproxime al máximo a la realidad del motor al que le estamos aplicando estas redes neuronales. Durante todo el capítulo se distinguirá para cada capítulo los ensayos realizados por ambos motores.

Todo el proceso de implementación de las algoritmos basados en redes neuronales artificiales han sido programados mediante el software MATLAB, para ello se han programado distintas funciones y scripts. A lo largo de las siguiente secciones se justificará la definición exacta del modelo, en base a las simulaciones realizadas con los distintos motores a partir de los ensayos realizados en el laboratorio.

3.1 Motor TJI

3.1.1 Parámetros a predecir y conjunto de datos

Durante el ensayo del motor TJI por parte del CMT, se obtuvieron distintas mediciones en los que se variaban una serie de parámetros. Estos parámetros eran el avance de la chispa (SA, Spark Advance), la apertura de la válvula encargada de regular el caudal de aire (Throttle), la duración de la inyección de combustible y las revoluciones por minuto del motor (rpm).

En el ensayo se realizaron cinco test, pero los test de interés fueron los tres últimos.

test 3	1350 RPM. Steps fueldur, SA y THR.
test 4	2000 RPM. Steps fueldur, SA y THR.
test 5	3000 RPM. Steps fueldur, SA y THR.

Tabla 3.1: Test laboratorio en motor TJI

De estos tres test el CMT realizó un post-procesado para obtener una serie de parámetros de interés para la combustión que se resumen en la siguiente tabla:

Nombre de la variable	Breve descripción
t	Variable tiempo
IMEP	Presión media efectiva indicada
maxdp	Valor máximo de la derivada de la presión en la cámara de combustión
CA10	Ángulo en el que se habrá producido el 10 % de la combustión
CA50	Ángulo en el que se habrá producido el 50 % de la combustión
CA90	Ángulo en el que se habrá producido el 90 % de la combustión
pint	Presión de admisión
maxp	Valor máximo de la presión en la cámara de combustión
SA	Avance de la chispa
SAdur	Duración del avance de la chispa
fueldur	Duración de la inyección del combustible
fuelpos	Ángulo en el que se inyecta el combustible
mair	Masa de aire que entra al pistón en un ciclo
lambda	Proporción aire combustible
EGRdes	Apertura de la válvula EGR deseada
EGRpos	Apertura real de la válvula EGR
THRdes	Apertura de la válvula Throttle deseada
THRpos	Apertura de real del válvula Throttle

Tabla 3.2: Datos proporcionados por el CMT del motor TJI

De estos parámetros, los más importante para ver que se produzca una correcta combustión son el IMEP, el CA10, el CA50 y el CA90.

El IMEP es la presión media efectiva indicada, este parámetro físicamente indica la presión teórica constante que supuestamente se ejerce durante cada carrera de potencia del motor para producir una potencia igual a la indicada.

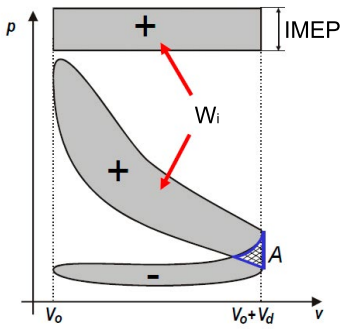


Figura 3.1: Diagrama P-V y representación de IMEP. Fuente: Cruz y col., 2002

En un motor TJI, que se encuentra en estudio, es frecuente que en algunos ciclos se inyecte combustible pero no se produzca combustión, el IMEP nos indica, de una forma indirecta, el trabajo efectivo, de tal forma que si tenemos un IMEP negativo o cercano al cero es porque o bien no se ha producido combustión (missfire) o bien se ha producido una combustión incompleta,

es decir que no se quema todo el combustible. Como se puede observar en la figura 3.2 hay algunos valores del IMEP que se encuentran muy cercanos al cero, por lo que en estos ciclos sea ha producido missfire. Sabiendo esto se ha generado una nueva variable llamada missfire, la cual valdrá 0 cuando no se produzca combustión y 1 cuando si que se produzca.

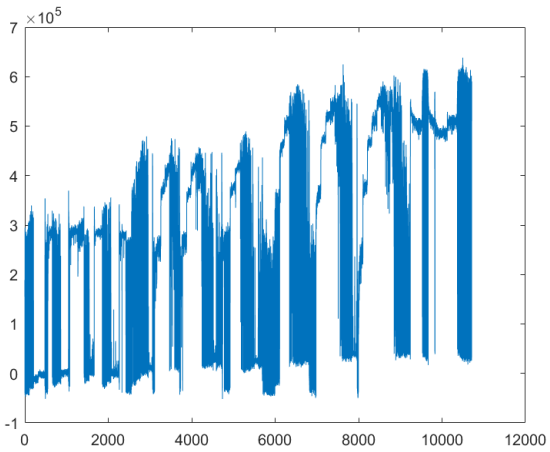


Figura 3.2: IMEP obtenido del test 3. Fuente: elaboración propia.

Dado este problema, se desarrollará un modelo que sea capaz de darnos una probabilidad de que se produzca missfire. Para ello se gastarán los datos de los test 3, 4 y 5 proporcionados por el CMT. Los inputs del modelo serán aquellos que se puedan medir durante el uso fuera del laboratorio del motor, quedando descartados de su uso el valor de la presión máxima en la cámara de combustión (maxp) y su derivada (maxdp), lo mismo ocurre con el CA10, CA50 y CA90, cuyos valores se calculan a partir del desarrollo de la presión en la cámara, una medición que no se puede realizar fuera del laboratorio. En la tabla 3.3 se resumen los inputs y outputs que se han gastado para predecir el missfire.

Inputs	Outputs
pint	missfire
SA	
fueldur	
mair	
EGRpos	
THRpos	

Tabla 3.3: Inputs y outputs de red para predecir missfire

Para terminar con este apartado, se va a explicar la distribución de los datos. Como nuestro principal objetivo es la adaptación de los datos, se va a reservar aproximadamente un 90 % de los datos a esta finalidad, del 10 % restante se reservará un 70 % al entrenamiento y un 30 % a la validación. El total del conjunto de datos uniendo los tres test de interés es de 34393 ciclos, por lo que se separarán los 4493 primeros y se mezclarán para que los datos de entrenamiento y validación sean similares; tras normalizar estos datos se separarán en 3456 ciclos para el entrenamiento y 1037 para la validación.

3.1.2 Estructura de la red

La estructura de la red es un parámetro determinante para reducir el error de nuestra red, actualmente no existe ningún método para determinar ni el número de capas ocultas ni el número de neuronas que se encuentran en estas, es por ello que para determinar la más adecuada para nuestro entrenamiento se han realizado una serie de pruebas, en función del método de entrenamiento, es por ello que vamos a separar el ELM de las redes neuronales entrenadas mediante el descenso del gradiente.

Redes neuronales

Para determinar la estructura de las redes neuronales entrenadas mediante el descenso del gradiente se van a realizar una serie de pruebas en las que se va a variar, tanto el número de capas ocultas como el número de neuronas en estas capas. Estas pruebas han sido realizadas todas con los misma función de activación, la función sigmoide (2.9); los mismos hiper-parámetros, $it_{max} = 5000$, $err_{min} = 0{,}001$ y learning rate $\eta = 0{,}1/32$; y el mismo método de entrenamiento, el mini batch-learning.

Como en este caso se trata de un problema de clasificación, en el que queremos determinar la probabilidad de que se produzca missfire, para evaluar a la red se contabilizaran cuantos aciertos y errores realiza. Es decir, si la probabilidad es mayor de un umbral que fijaremos, y además, no se produce missfire se contará como acierto; lo mismo ocurrirá cuando la probabilidad sea menor que este umbral y se produzca missfire. Como error se contabilizarán los casos en los que no se de esto, es decir que se espere missfire y el modelo prediga que no y viceversa; este umbral se ha fijado en 0,1, es decir que por debajo del 10 % de probabilidad se considerará que se produce missfire. Cabe recordar que la salida será un 1 cuando no se produzca missfire y un 0 cuando si que se produzca. Además también es de gran interés conocer el tiempo que tarda en entrenarse la red, ya que cuando se realicen un mayor número de iteraciones, las diferencias en el tiempo de ejecución que ahora aparentan despreciables, se pueden convertir en diferencia de horas de ejecución.

Los datos de entrada y de salida siempre serán los mismos, por lo que la capa de entrada y de salida se mantendrán constantes durante todas las pruebas, siendo 6 y 1 respectivamente. En cuanto a las estructuras que se han realizado para las pruebas son las siguientes:

- Prueba 1: una capa oculta con 30 neuronas $[6, 30, 1]$.
- Prueba 2: una capa oculta con 20 neuronas $[6, 20, 1]$.
- Prueba 3: dos capas ocultas con 20 neuronas cada una $[6, 20, 20, 1]$.
- Prueba 4: dos capas ocultas con 10 neuronas cada una $[6, 10, 10, 1]$.
- Prueba 5: tres capas ocultas con 10 neuronas cada una $[6, 10, 10, 10, 1]$.
- Prueba 6: tres capas ocultas con 7 neuronas cada una $[6, 7, 7, 7, 1]$.

y los resultados obtenidos se recogen en la siguiente tabla:

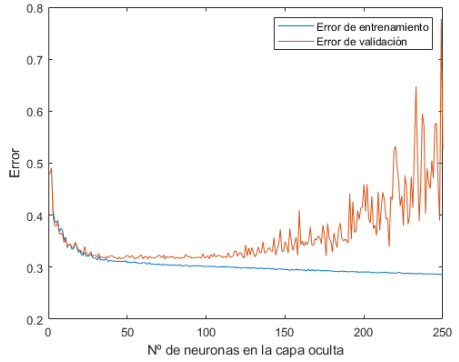
Prueba	1	2	3	4	5	6
Estructura de la red	$[6, 30, 1]$	$[6, 20, 1]$	$[6, 20, 20, 1]$	$[6, 10, 10, 1]$	$[6, 10, 10, 10, 1]$	$[6, 7, 7, 7, 1]$
Error entrenamiento	$0,3005$	$0,3034$	$0,2958$	$0,3017$	$0,2947$	$0,3085$
Error validación	$0,3412$	$0,3403$	$0,3364$	$0,3368$	$0,3493$	$0,3365$
Aciertos	807	806	810	803	820	802
Errores	230	231	227	234	217	235
Tiempo de entrenamiento (s)	189,34	188,01	200,56	196,4	207,46	188,56

Tabla 3.4: Resultados de pruebas de estructura de la red para las redes neuronales entrenadas con el descenso del gradiente para la predicción del missfire. Fuente: elaboración propia.

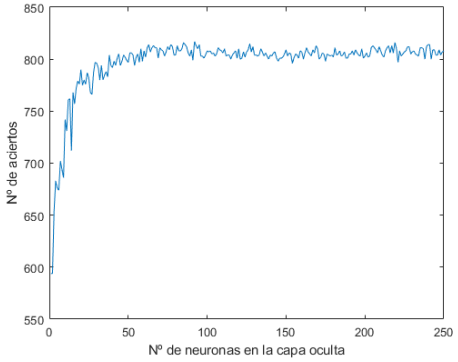
A la vista de los resultados de la tabla se pueden extraer una serie de conclusiones. La primera de ellas es entender los valores de errores de entrenamiento y validación, estos miden el error en la probabilidad de que se produzca el missfire, es decir el error entre la salida real cuyos valores serán $\{0, 1\}$ y los valores de la salida del modelo que estarán comprendidos entre $(0, 1)$, este error nos sirve para entrenar el modelo, pero para nosotros tiene un valor mucho más significativos medir los aciertos y errores de nuestra red neuronal, de tal forma que los datos de aciertos y errores serán mucho más relevantes que los de los errores de entrenamiento y validación. Teniendo esto en cuenta, observamos que en la prueba 5, con estructura $[5, 10, 10, 10, 1]$, se ha obtenido un valor más elevado de aciertos comparado con el resto, a coste de tener un mayor tiempo de ejecución por la complejidad de la red; por lo que de ahora en adelante se trabajará con esta estructura.

Extreme Learning Machine

En cuanto al Extreme Learning Machine, lo único que varía en su modelado es el número de neuronas en la capa oculta. Para evaluar la mejor opción se ha entrenado y validado la red con distinto número de neuronas. Como en las redes neuronales entrenadas mediante el descenso del gradiente, nuestro mayor interés es el que el número de aciertos sea el máximo, pero el error nos sirve también para estudiar a nuestro modelo.



(a) Variación del error en función del número de neuronas en la capa oculta para la predicción del missfire.



(b) Variación del nº de aciertos en función del número de neuronas en la capa oculta.

Figura 3.3: Resultados de pruebas para determinar el número de neuronas en la capa oculta para la predicción del missfire. Fuente: elaboración propia.

En la figura 3.3a se puede observar como a medida que se aumenta el número de neuronas se reduce el error de entrenamiento pero aumenta considerablemente el error de validación a partir de un cierto punto, esto es debido a un fenómeno denominado como overfitting, explicado anteriormente en 2.10. Como se puede ver también en la figura 3.3b, el número de aciertos se mantiene, más o menos, constante a partir de un cierto punto. Nuestro principal objetivo es aumentar el número de aciertos pero entrenando de manera adecuada, es por esto que elegiremos el número de neuronas de la capa oculta, no tanto en función del número de aciertos si no en un error de validación mínimo. Para ello tomaremos como valor adecuado un número de 80 neuronas en la capa oculta, de forma que la estructura de la red será $[6, 80, 1]$. Para poder comparar con las redes neuronales entrenadas mediante el descenso del gradiente se ha calculado el tiempo de entrenamiento y el número de aciertos para esta estructura, y se recogen en la tabla 3.5

Estructura de la red	$[6, 80, 1]$
Error de entrenamiento	0,3044
Error de validación	0,3238
Nº de aciertos	808
Nº de errores	229
Tiempo de entrenamiento (s)	0,0128

Tabla 3.5: Resultados de pruebas de EML estando la capa intermedia formada por 70 neuronas para el ensayo de motor TJI

3.1.3 Función de activación

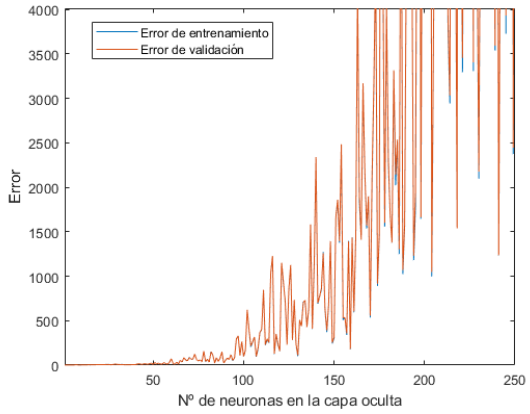
Redes neuronales

Como se vio en la sección 2.2, existen diversas funciones de activación que se pueden utilizar para el uso de nuestras redes neuronales. Como vimos, para el entrenamiento de nuestra red mediante el uso del algoritmo de backpropagation (2) es necesaria la derivada de nuestra función de activación, es por eso que solo valoraremos funciones de activación que sean derivables en todo el rango. Este problema se puede resolver utilizando otros algoritmos, pero no se va a entrar en la definición de ellos puesto que no es el principal objetivo del trabajo.

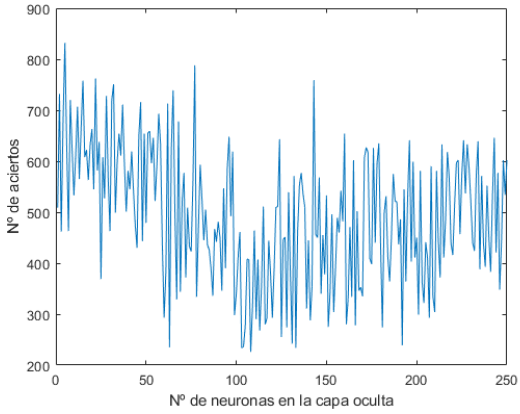
Particularizando para el objetivo de estudio de este motor, en el que se trata de predecir si se va a predecir missfire durante el proceso de combustión, el uso de una red neuronal, cuya salida esté comprendida entre $(0, 1)$ nos indicará el porcentaje de que se produzca missfire. Por estas razón y para mantener al estabilidad del algoritmo backpropagation, la función de activación deberá ser derivable y su salida acotada entre $(0, 1)$, por lo que de las nombradas en el capítulo 2.2 la función sigmoide (2.9) es la única que se adecua a estas características y que por lo tanto aplicaremos.

Extreme Learning Machine

Para el estudio del caso del ELM se van a comparar los resultados para distintas neuronas en la capa intermedia comparando la función tangente hiperbólica y la función sigmoide, que son las funciones más utilizadas. Aprovechando los resultados obtenidos en el apartado anterior, que se pueden observar en la figura 3.3, se van a extraer las mimas gráficas pero usando la función tangente hiperbólica (2.11).



(a) Variación del error en función del número de neuronas en la capa oculta para la predicción del missfire usando la función de activación tangente hiperbólica.



(b) Variación del nº de aciertos en función del número de neuronas en la capa oculta usando la función de activación tangente hiperbólica.

Figura 3.4: Resultados de pruebas para determinar el número de neuronas en la capa oculta para la predicción del missfire usando la función tangente hiperbólica. Fuente: elaboración propia.

Como se puede observar en las gráficas 3.4, el uso de la función tangente hiperbólica como función de activación implica una gran inestabilidad en los resultados y un error mucho mayor que los obtenidos usando la función sigmoide. Por lo tanto, de ahora en adelante se gastará únicamente la función de activación sigmoide para el uso del ELM en el motor TJI.

3.1.4 Hiper-parámetros

Como se comentó anteriormente, en el apartado 2.3.3, existen una serie de parámetros que determinan el proceso de entrenamiento de la red, a estos se les denomina hiper-parámetros porque son parámetros ajustables que permiten controlar el proceso de entrenamiento. El número de capas ocultas y el número de neuronas, así como la función de activación de nuestra red neuronal pueden considerarse hiper-parámetros también, pero en este estudio se han considerado a parte para dotarles de una mayor relevancia en este trabajo.

En este apartado se van a estudiar tres parámetros, estos parámetros son el learning rate (η), el número de iteraciones máximas (it_{max}) y el error mínimo ($error_{min}$). Para empezar, el error mínimo y el número de iteraciones máximas serán los valores que determinarán el número de iteraciones que se van a realizar en nuestro entrenamiento, la red se mantendrá en el proceso de entrenamiento hasta que o bien, el número de iteraciones que lleva el proceso alcance el número de iteraciones máximas, o bien el error obtenido por la red sea el error mínimo. Estos hiper-parámetros definirán la precisión de la red al terminar el entrenamiento, puesto que un mayor número de iteraciones implicará que nos acerquemos a un mínimo en nuestra función

de costes; también determinarán la velocidad del entrenamiento dada la estrecha relación que mantienen la velocidad de entrenamiento con el número de iteraciones que se realicen. Por lo tanto, estos dos hiper-parámetros son parámetros que se variarán en función de la precisión del entrenamiento que deseemos, por lo general, la condición de que el entrenamiento finalice cuando se obtenga un error mínimo no se va a cumplir puesto que, con los datos de los que disponemos y la capacidad de computación disponible, no podemos alcanzar un modelo que se ajuste tan bien a la realidad. En cuanto al número de iteraciones máximas, ya se ha variado para realizar distintas pruebas anteriormente, y se ajustará más adelante en función de la precisión de los resultados que queramos obtener.

Un parámetro muy distintos a los dos comentados anteriormente es el learning rate, de este parámetro va a depender la velocidad y la precisión de la convergencia en un mínimo de nuestra función de coste. Un learning rate demasiado elevado puede implicar saltos demasiado grandes al descender el gradiente, lo cual induciría inestabilidad en el entrenamiento teniendo problemas para converger adecuadamente y pese a parecer una opción que fuera mejor, empeoraría el proceso de entrenamiento. Por el contrario, un valor del learning rate excesivamente pequeño implicaría un proceso de iteración excesivamente lento que consumiría demasiado tiempo. Para determinar el valor adecuado con nuestro modelo se han realizado una serie de pruebas en las que se ha variado este parámetro para ver su funcionamiento y como afecta a los resultados. En concreto se han realizado seis pruebas, con la estructura y la función de activación definida anteriormente, usando el mini batch-learning y variando el learning rate entre los siguientes valores:

- Learning rate $\eta = \frac{2}{length_{batch}}$.
- Learning rate $\eta = \frac{1}{length_{batch}}$.
- Learning rate $\eta = \frac{0{,}5}{length_{batch}}$.
- Learning rate $\eta = \frac{0{,}1}{length_{batch}}$.
- Learning rate $\eta = \frac{0{,}05}{length_{batch}}$.
- Learning rate $\eta = \frac{0{,}01}{length_{batch}}$.

El parámetro $length_{batch}$ sirve para normalizar el learning rate entre los datos que va a tomar la red para el entrenamiento, como se trata del entrenamiento mini batch-learning, se tomará $length_{batch} = 32$. Obteniendo los siguientes resultados:

Prueba	1	2	3	4	5	6
Learning rate	$\eta = \frac{2}{length_{batch}}$	$\eta = \frac{1}{length_{batch}}$	$\eta = \frac{0{,}5}{length_{batch}}$	$\eta = \frac{0{,}1}{length_{batch}}$	$\eta = \frac{0{,}05}{length_{batch}}$	$\eta = \frac{0{,}01}{length_{batch}}$
Error entrenamiento	$0,3049$	$0,2724$	$0,3124$	$0,3243$	$0,3575$	$0,3767$
Error validación	$0,3440$	$0,3806$	$0,3287$	$0,3368$	$0,3497$	$0,3643$
Aciertos	792	793	798	800	801	715
Errores	245	244	239	237	236	322
Tiempo de entrenamiento (s)	863,63	897,11	898,26	890,87	899,25	867,77

Tabla 3.6: Resultados de pruebas de learning rate para las redes neuronales entrenadas con el descenso del gradiente para la predicción del missfire. Fuente: elaboración propia.

Como se puede observar, en la tabla 3.6, en las pruebas 3,4 y 5 los resultados en los aciertos y errores son muy parecidos, es por esto que vamos a fijarnos, no solo en los aciertos y errores, si

no también en el tiempo de entrenamiento, escogiendo por tanto de estas tres pruebas la prueba 4, que tiene un tiempo de ejecución 10 segundos menor que el resto de opciones. Por lo tanto de ahora en adelante, tomaremos como learning rate el valor $\eta = \frac{0{,}1}{length_{batch}}$.

3.2 Motor diésel con EGR de baja presión

3.2.1 Parámetros a predecir y conjunto de datos

En el caso de este motor, se realizó el ensayo de un ciclo NEDC (New European Driving Cycle o Nuevo Ciclo Europeo de Conducción), una prueba que se realizaba anteriormente para evaluar el impacto medioambiental de los vehículos, necesarios para homologar cifras de consumo y emisiones de gases. Este ciclo está basado en cuatro ciclos urbanos que se repiten y un ciclo extra-urbano, en la figura 3.5 se pueden distinguir claramente estos cinco ciclos.

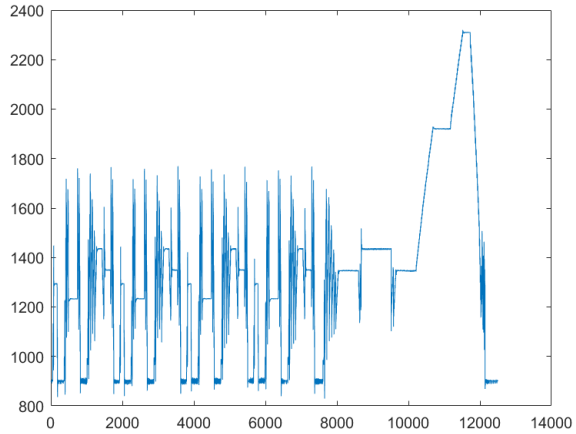


Figura 3.5: RPM del ciclo NEDC. Fuente: elaboración propia.

De este ensayo se obtuvieron una gran cantidad de variables provenientes de la ECU, la unidad de control del motor. Además también se midió la evolución de la presión en la cámara. El total del ensayo está formado por 12501 mediciones de todas las medidas, siendo este el número de datos disponibles para realizar el entrenamiento, la validación y la adaptación. Como la finalidad del trabajo es el desarrollo de las redes neuronales adaptativas el gran grueso del ensayo se ha separado para el proceso de adaptación, de tal forma que se han tomado 896 ciclos para el entrenamiento, 300 para la validación y 11305, el resto, para el proceso de adaptación. Los datos de entrenamiento y validación se han tomado del principio del ensayo y han sido mezclados para que sean uniformes, posteriormente han sido normalizados. La normalización también ha sido realizada para el proceso de adaptación, pero no han sido mezclados puesto que queremos que simule un ciclo real, con sus partes transitorias.

Así como en el motor TJI, el CMT proporcionó los datos ya procesados, en este caso se van a procesar los datos para obtener el CA10, CA50, CA90 y rendimiento de cada pistón. Se ha de tener en cuenta que durante el ensayo no se realizaron adecuadamente las medidas de la presión en el tercer cilindro, obteniendo unas presiones que no corresponden a la realidad, como se puede observar en la figura 3.6.

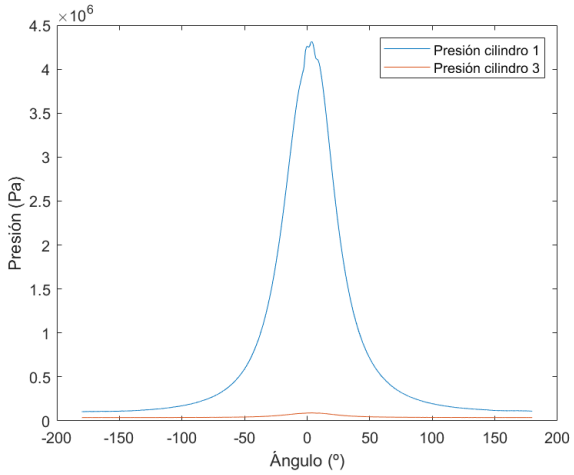


Figura 3.6: Presión del primer y tercer cilindro en un ciclo. Fuente: elaboración propia.

El cálculo de estas variables es de gran interés para evaluar como se realiza la combustión. El rendimiento es un parámetro de gran interés y que se busca maximizar siempre para poder obtener en el eje la mayor cantidad de energía procedente del combustible. Por otra parte, hay estudios experimentales (Klimstra, 1985) que relacionan combustiones óptimas con un CA50 situado en torno a siete u ocho grados después del punto muerto superior (PMS)

Cálculo del CA10, CA50 y CA90

El CA50 es el ángulo en el que se habrá producido la mitad de la combustión. El cálculo del CA10, CA50 y CA90 es muy similar dado que una vez tengamos como evoluciona la combustión ya solo habrá que variar el porcentaje que buscamos. La evolución de esta combustión es el Heat Release (HR), esto es la integral del desarrollo del calor liberado por el combustible, y que proviene de integrar el Heat Release Rate (HRR), que es el desarrollo del calor liberado por el combustible. Para el cálculo existen distinto métodos de cálculo, pero como se propuso en B.Pla y col., 2020, se aplicará la siguiente fórmula:

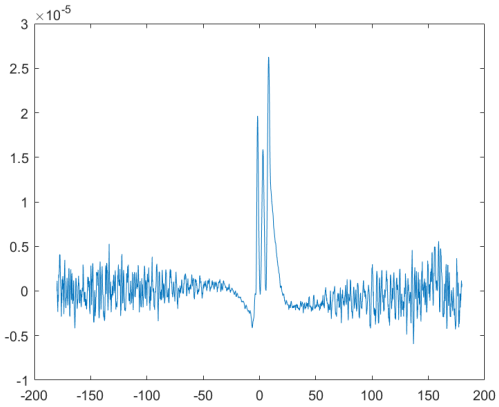
$$HRR = \frac{\kappa}{\kappa - 1} pdV + \frac{1}{\kappa - 1} Vdp \tag{3.1}$$

siendo κ un parámetro que se determina experimentalmente y que se tomará como 1,3.

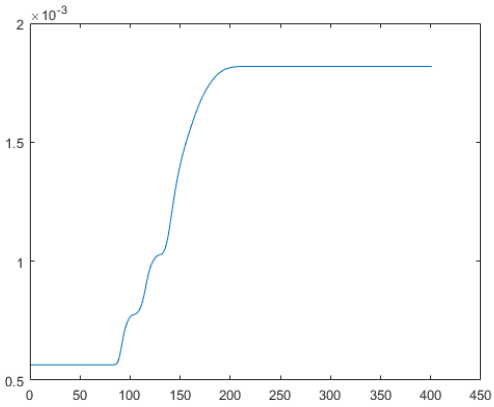
Una vez se tiene el HRR, para el cálculo del CAx bastará con integrar el HRR:

$$\int^{\alpha=CA_x} HRRd\alpha = \frac{xHR}{100} \tag{3.2}$$

siendo x el porcentaje del cual queremos calcular el CAx, por ejemplo, en el caso de querer calcular el CA50, $x = 50$.



(a) Hear Release Rate (HRR)



(b) Heat Release (HR)

Figura 3.7: HRR y HR de un ciclo del ensayo. Fuente: elaboración propia.

Cálculo del rendimiento

El cálculo del rendimiento del ciclo resulta más sencillo que el cálculo anterior, para calcular el rendimiento bastará con dividir el IMEP, comentado anteriormente en el apartado 3.1.1, entre la masa de combustible inyectada por pistón y cilindro.

$$rend = \frac{IMEP}{m_{fuel}} \tag{3.3}$$

El IMEP es la presión media efectiva indicada y se calcula como la división entre el trabajo indicado y el volumen desplazado.

$$IMEP = \frac{W_i}{\Delta V} \tag{3.4}$$

siendo W_i el trabajo indicado del ciclo y que se calcula de la siguiente forma:

$$W_i = \int PdV \tag{3.5}$$

Una vez ya tenemos el cálculo del CA10, CA50, CA90 y rendimiento para cada pistón ya se puede formar la estructura de datos para el modelado de la red neuronal. El objetivo de este modelo es poder calcular el CA10, CA50, CA90 y rendimiento para cada cilindro. Como no tenemos datos disponibles del tercer cilindro, tendremos tres para cada uno de estos. En cuanto a las entradas de la red, de todos los datos disponibles en le ECU se han seleccionado los siguientes parámetros:

- El grado de carga (α).
- La presión de admisión (P_{adm}).
- La temperatura de admisión (T_{adm}).
- La masa de combustible inyectada por pistón y ciclo (m_{fuel})
- Las revoluciones por minuto (RPM)

- La temperatura de escape (T_{esc})
- La masa de aire en la admisión (m_{air})
- El porcentaje de apertura de la válvula EGR de baja presión (EGR_{LOW})

De tal forma que los inputs y outputs se pueden resumir en la siguiente tabla:

Inputs	Outputs
α	$CA10_{cil1}$
P_{adm}	$CA10_{cil2}$
T_{adm}	$CA10_{cil4}$
m_{fuel}	$CA50_{cil1}$
RPM	$CA50_{cil2}$
T_{esc}	$CA50_{cil4}$
m_{air}	$CA90_{cil1}$
EGR_{LOW}	$CA90_{cil2}$
	$CA90_{cil4}$
	$rend_{cil1}$
	$rend_{cil2}$
	$rend_{cil4}$

Tabla 3.7: Inputs y outputs de red para motor EGR de baja presión

3.2.2 Estructura de la red

La estructura de la red es un parámetro muy determinante, sobre todo en el caso de las redes neuronales entrenadas mediante el descenso del gradiente y no tanto en el caso del EML. Es por eso que se han realizado una serie de pruebas determinar la estructura a realizar.

Redes neuronales

Para determinar la estructura de la red de las redes neuronales entrenadas mediante el descenso del gradiente, se van a realizar unas pruebas con distintas estructuras con los datos de entrenamiento y validación, en concreto se realizarán seis pruebas. Todas ellas tienen los mismos hiper-parámetros, $it_{max} = 50000$, $err_{min} = 0{,}001$ y learning rate $\eta = 0{,}1/32$; la misma función de activación (la función tangente hiperbólica 2.11) y el método de entrenamiento (mini-batch learning).

De estas pruebas resulta de interés conocer parámetros como el error de entrenamiento, el error de validación y los tiempos de ejecución. Un error de entrenamiento pequeño es interesante, pero carece de sentido si no se compara con el de validación, puesto que un error de validación muy grande frente a uno de entrenamiento pequeño indica la presencia de overfitting.

Los datos de entrada y de salida de las pruebas son los comentados anteriormente, por lo que la capa de entrada y salida de la red siempre será la misma, ocho y doce respectivamente; lo que se ha variado ha sido el número de capas ocultas y las neuronas de esta de la siguiente manera:

- Prueba 1: una capa oculta con 30 neuronas $[8, 30, 12]$.
- Prueba 2: una capa oculta con 20 neuronas $[8, 20, 12]$.
- Prueba 3: dos capas ocultas con 20 neuronas cada una $[8, 20, 20, 12]$.
- Prueba 4: dos capas ocultas con 10 neuronas cada una $[8, 10, 10, 12]$.
- Prueba 5: tres capas ocultas con 10 neuronas cada una $[8, 10, 10, 10, 12]$.
- Prueba 6: tres capas ocultas con 7 neuronas cada una $[8, 7, 7, 7, 12]$.

Los resultados obtenidos se resumen en la siguiente tabla:

Prueba	1	2	3	4	5	6
Estructura de la red	$[8, 30, 12]$	$[8, 20, 12]$	$[8, 20, 20, 12]$	$[8, 10, 10, 12]$	$[8, 10, 10, 10, 12]$	$[8, 7, 7, 7, 12]$
Error entrenamiento	$0,0752$	$0,0763$	$0,0667$	$0,0748$	$0,0694$	$0,0748$
Error validación	$0,0548$	$0,0518$	$0,0609$	$0,054$	$0,0538$	$0,0525$
Error validación % CA10	$17,7393$	$16,1584$	$19,5369$	$16,155$	$15,9479$	$15,9133$
Error validación % CA50	$8,4057$	$7,9888$	$8,8794$	$7,262$	$7,9115$	$7,7138$
Error validación % CA90	$6,6556$	$6,5163$	$6,378$	$6,1126$	$5,9789$	$5,7732$
Error validación % rendimiento	$6,0987$	$6,1660$	$6,1732$	$6,2891$	$6,2589$	$6,8741$
Tiempo de entrenamiento (s)	1891	1834	1993	1968	2094	1930

Tabla 3.8: Resultados de pruebas de estructura de la red para las redes neuronales entrenadas con el descenso del gradiente en el motor diésel con EGR de baja presión. Fuente: elaboración propia.

Como se puede apreciar en la tabla, un mayor número de neuronas y capas ocultas no tiene porque significar un menor error ni un menor tiempo de entrenamiento. Es por eso que se realizan estas pruebas, para determinar adecuadamente la estructura de la red. Por otra parte, también es muy importante la velocidad de entrenamiento, en este caso, la red se ha entrenado durante tan solo 50000 iteraciones, pero para obtener mayor precisión se podrían haber realizado más iteraciones, aumentando considerablemente el tiempo de entrenamiento.

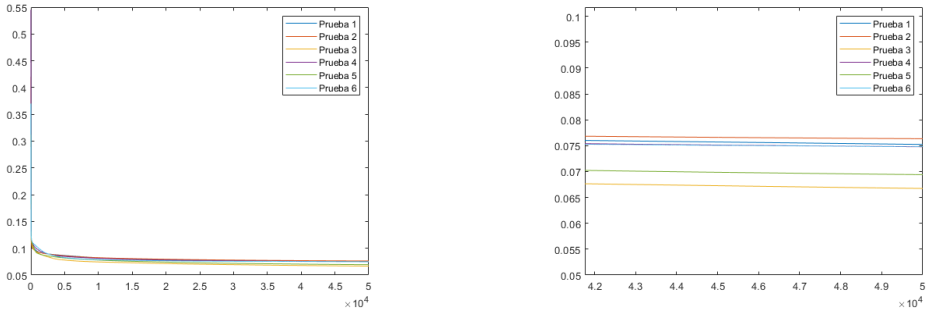


Figura 3.8: Variación error en función del número de iteraciones. Fuente: elaboración propia

Con los resultados anteriores de la tabla 3.8 se puede extraer que pese a que la estructura de la tercera prueba presenta un menor error de entrenamiento, esta presenta un elevado error de validación en comparación con el resto de estructuras; y que la segunda prueba, con estructura $[8, 20, 12]$ es la que mejor resultados de validación y mejor tiempo de entrenamiento tiene. Por lo que se toma la estructura $[8, 20, 12]$ como la mejor estructura, para el uso de redes neuronales

entrenadas mediante el descenso del gradiente, y se gastará de ahora en adelante para definir el resto de parámetros.

Extreme Machine Learning

Para el Extreme Machine Learning, el único parámetro a ajustar, a parte del proceso de adaptación, es el número de neuronas de la capa oculta. Para determinar esto se ha realizado un ensayo en el que se ha variado el número de neuronas y se ha calculado la media del error de la salida, esto se recoge en la gráfica 3.9.

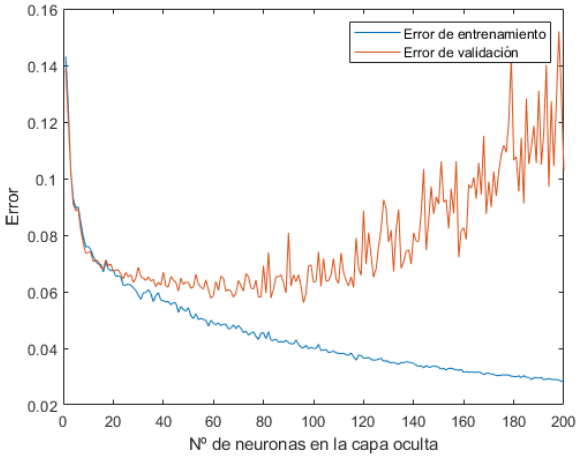


Figura 3.9: Error de entrenamiento y validación en función del número de neuronas. Fuente: elaboración propia.

Viendo esta gráfica se puede observar con claridad que conforme aumentamos el número de neuronas en la capa intermedia, el error de entrenamiento disminuye considerablemente pero el error de validación aumenta significativamente. Esto es debido a que se produce el fenómeno conocido como overfitting (2.10), por lo que el número de neuronas más apropiado será aquel en el que el error de validación sea mínimo,esto se da cuando se tienen 70 neuronas en la capa intermedia. Para poder compararlo con el entrenamiento de las redes neuronales mediante el descenso del gradiente se han calculado los mismos resultados, que se recogen en la siguiente tabla:

Estructura de la red	$[8, 70, 12]$
Error de entrenamiento	0,0466
Error de validación	0,0613
Error validación % CA10	24,8418
Error validación % CA50	8,0078
Error validación % CA90	7,8349
Error validación % rendimiento	5,4928
Tiempo de entrenamiento (s)	0,0043

Tabla 3.9: Resultados de pruebas de EML estando la capa intermedia formada por 70 neuronas

En la tabla 3.9 se puede observar la escasa velocidad de entrenamiento del Extreme Machine Learning, esto implica un mayor error en la validación comparado con las redes neuronales entrenadas mediante el descenso del gradiente, que tienen un error de validación menor.

3.2.3 Función de activación

Redes neuronales

Al contrario que en el motor TJI, en la aplicación de las redes neuronales para este motor, al ser un problema de regresión y no de clasificación no es necesario que la salida de la función de activación esté comprendida entre $(0, 1)$. Lo que si es fundamental para dotar de robustez al algoritmo de backpropagation es el uso de una función de activación que sea derivable en todo su dominio y que la función de activación tenga un rango de salida acotado. De las estudiadas en el apartado 2.2, las funciones de activación que son derivables en todo su dominio y además tienen un rango acotado son la función sigmoide (2.9) y la función tangente hiperbólica (2.11). Para determinar cual de ellas se comporta mejor se va a realizar, una prueba con la función sigmoide y otra con la función tangente hiperbólica, el resto de parámetros van a ser constantes para las dos, siendo la estructura de la red $[8, 20, 12]$, las iteraciones máximas $it_{max} = 10000$, el error mínimo $err_{min} = 0{,}001$, el learning rate $\eta = 0{,}1/32$ y el método de entrenamiento será el mini batch-learning.

Los resultados obtenidos se recogen en la siguiente tabla:

Función de activación	Tangente hiperbólica	Sigmoide
Estructura de la red	$[8, 20, 12]$	$[8, 20, 12]$
Error de entrenamiento	0,0516	0,0698
Error de validación	0,0826	0,0977
Error validación % CA10	22,7268	28,5349
Error validación % CA50	13,6161	24,4471
Error validación % CA90	9,1602	10,3658
Error validación % rendimiento	5,4755	6,1357
Tiempo de entrenamiento (s)	394,35	362,3460

Tabla 3.10: Resultados de pruebas para determinar la función de activación en el motor diésel con EGR de baja presión

A la vista de estos datos es fácil afirmar que los resultados del entrenamiento con la tangente hiperbólica son considerablemente mejores que los obtenidos con la función de sigmoide, es por esto que de ahora en adelante se utilizará la función de activación tangente hiperbólica con la intención de que el error sea menor.

Extreme Learning Machine

En cuanto a la función de activación utilizada en el ELM, al igual que se ha realizado en el apartado 3.1.3, se va a calcular como varía el error, tanto de entrenamiento como de validación, cuando se utiliza la función de activación tangente hiperbólica y así poder compararlo con la figura 3.9.

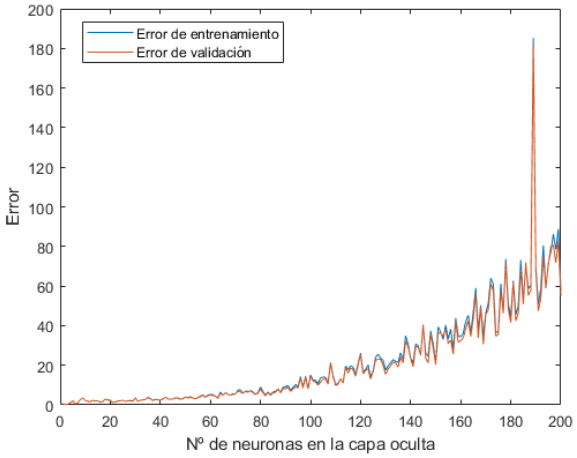


Figura 3.10: Error de entrenamiento y validación en función del número de neuronas con función de activación tangente hiperbólica. Fuente: elaboración propia.

Como se puede observar en la figura 3.10 y comparándola con la figura 3.9, el error obtenido utilizando la función sigmoide es mucho menor, y se comporta de una forma muy estable. Por lo tanto, se tomará la función de activación sigmoide para el ELM como la función de activación que produce un error menor.

3.2.4 Hiper-parámetros

Al igual que ocurría en el apartado 3.1.4, los valores del número de iteraciones máximos y el error mínimo los variaremos en función de la precisión del entrenamiento que pretendamos obtener, por lo que no es de especial relevancia definirlos. Lo contrario ocurre con el learning rate, al igual que en el apartado comentado anteriormente, se van a realizar una serie de pruebas variando este valor con el fin de determinar cual nos proporciona menor error y mayor estabilidad en el entrenamiento, y mantiniendo el resto de los parámetros que se habían definido hasta ahora, las pruebas se han realizado con 20000 iteraciones y mediante el entrenamiento de tipo mini batch-learning. Por lo tanto, se han realizado distintas pruebas variando el learning rate entre los siguiente valores:

- Learning rate $\eta = \frac{1}{length_{batch}}$.
- Learning rate $\eta = \frac{0{,}5}{length_{batch}}$.
- Learning rate $\eta = \frac{0{,}1}{length_{batch}}$.
- Learning rate $\eta = \frac{0{,}05}{length_{batch}}$.
- Learning rate $\eta = \frac{0{,}01}{length_{batch}}$.

- Learning rate $\eta = \frac{0{,}005}{length_{batch}}$.

siendo el $length_{batch}$ un parámetro característico del mini batch-learning, el cual se encarga de normalizar los datos entre el número de datos, en nuestro caso $length_{batch} = 32$.

En cuanto a los resultados, se recogen en la siguiente tabla:

Prueba	1	2	3	4	5	6
Learning rate	$\eta = \frac{1}{length_{batch}}$	$\eta = \frac{0{,}5}{length_{batch}}$	$\eta = \frac{0{,}1}{length_{batch}}$	$\eta = \frac{0{,}05}{length_{batch}}$	$\eta = \frac{0{,}01}{length_{batch}}$	$\eta = \frac{0{,}005}{length_{batch}}$
Error entrenamiento	$0,0852$	0,0806	$0,0783$	$0,0826$	$0,0964$	$0,0987$
Error validación	$0,0710$	$0,060$	$0,0539$	$0,0561$	$0,0680$	$0,0681$
Error validación % CA10	39,4	34,84	24,13	25,739	29,243	28,877
Error validación % CA50	23,97	11,459	18,637	14,55	21,902	21,403
Error validación % CA90	26,299	18,468	19,55	20,881	25,767	25,505
Error validación % rendimiento	7,445	6,8401	5,3592	5,6275	5,911	6,008
Tiempo de entrenamiento (s)	803,42	794,11	794,20	796,05	808,69	809,88

Tabla 3.11: Resultados de pruebas de learning rate para las redes neuronales entrenadas con el descenso del gradiente para el motor diésel con EGR de baja presión. Fuente: elaboración propia.

Viendo los resultados obtenidos en la tabla, podemos afirmar que un learning rate $\eta = \frac{0{,}1}{length_{batch}}$, realizado en la prueba 3, es el que mejor resultado nos aporta, por lo que de ahora en adelante se gastará este parámetro para entrenar esta red neuronal.

Capítulo 4

Adaptación del modelo

A lo largo de este capítulo se tratará el proceso de adaptación de cada modelo, definiendo como va a ser este proceso de adaptación y los parámetros que lo definen.

Hasta el momento, se ha definido el método de entrenamiento mediante el uso de los datos de entrenamiento y validación. Pero la parte más innovadora de este trabajo se encuentra en la adaptación del modelo conforme se va usando, esto implica que el modelo podrá adaptarse a la evolución del motor y sus elementos debido, por ejemplo, al envejecimiento de este. Para ello se van a utilizar distintas técnicas en función del tipo de red neuronal que se tenga.

4.1 Redes neuronales

Para empezar, comenzaremos con las redes neuronales cuyo entrenamiento está basado en el descenso del gradiente. Durante todo el desarrollo del entrenamiento se ha utilizado el método de entrenamiento mini batch-learning, dado que es un método de entrenamiento estable y más rápido que otros como el online learning. El online learning es frecuentemente utilizado cuando los datos se producen durante el uso de la red, por esto lo utilizaremos para el proceso de adaptación. A partir de lo definido en los apartados anteriores, entrenaremos una red neuronal que se adapte correctamente a los datos de entrenamiento y validación y, posteriormente, la iremos adaptando a los nuevos datos mediante el online learning, de tal forma que irá ajustándose y mejorando para cada ciclo, de tal forma que sea capaz de predecir correctamente los siguientes datos. Sobre estos datos se podrán iterar las veces que se desee, de tal forma que por cada iteración se reducirá el error.

El objetivo es que una vez esté adaptada sea capaz de predecir correctamente los nuevos ciclos, sin perder una gran precisión en los datos para los cuales se ha entrenado. Con el fin de ver su comportamiento entrenaremos un modelo con los datos de entrenamiento, función de activación, estructura de la red e hiper-parámetros definidos anteriormente para cada modelo durante 50000 iteraciones, para obtener un modelo preciso que se comporte adecuadamente con los datos de entrenamiento y validación, y calcularemos como se comporta con los datos de adaptación. Posteriormente se realizará el proceso de adaptación mediante el online learning, iterando 100 veces sobre cada dato y comprobaremos como se comporta con los datos de entrenamiento, valida-

ción y adaptación para poder sacar conclusiones sobre como ha mejorado, o empeorado, nuestro modelo.

4.1.1 Motor TJI

Como se ha descrito anteriormente, para realizar el proceso de adaptación se ha pre-entrenado una red con los datos de entrenamiento y posteriormente realizada la adaptación a los nuevos datos. Al tratarse de un problema de clasificación, se valorará más el número de aciertos que el error de la red. Como en este caso los datasets tienen distintos tamaños, se ha normalizado en número de aciertos dividiéndolo entre el tamaño del dataset, de tal forma, que un 100 % de precisión implica que no se ha cometido ningún error en la predicción. También se ha medido el tiempo de entrenamiento y validación para compararlo más adelante con otros métodos de entrenamiento. Los resultados obtenidos se recogen en la siguiente tabla:

	Entrenamiento inicial	Adaptación
Error datos entrenamiento	$0,3111$	$0,5707$
Error datos validación	$0,3189$	$0,5620$
Error datos adaptación	0,7098	0,0,4077
Error todos los datos	0,6847	0,4139
Precisión datos entrenamiento (%)	78,36	57,12
Precisión datos validación (%)	77,92	60,08
Precisión datos adaptación (%)	55,15	81,76
Precisión todos los datos (%)	56,54	79,22
Tiempo de entrenamiento (s)	2067	3474

Tabla 4.1: Comparación de resultados entre entrenamiento normal y adaptación del modelo para los datos del motor TJI. Fuente: elaboración propia.

Como se puede observar en la tabla 4.1, el modelo pre-entrenado obtiene casi un 80 % de precisión para los datos de entrenamiento y validación, pero la precisión cae bruscamente hasta casi el 50 % en los nuevos datos de adaptación. Por el contrario, tras el proceso de adaptación, la precisión de aciertos en entrenamiento y validación decae hasta el 60 % aproximadamente, pero la precisión de los nuevos datos es mucho mayor que la precisión de únicamente el entrenamiento inicial.

El interés del proceso de adaptación reside en una mayor precisión en los datos de adaptación, puesto que para los datos de entrenamiento y validación, se gasta el modelo pre-entrenado y conforme obtenemos nuevos datos se realiza la adaptación. De esta forma, se entiende la importancia de la adaptación del modelo, siendo capaz de mejorar con el paso del tiempo y la obtención de nuevos datos.

4.1.2 Motor diésel con EGR de baja presión

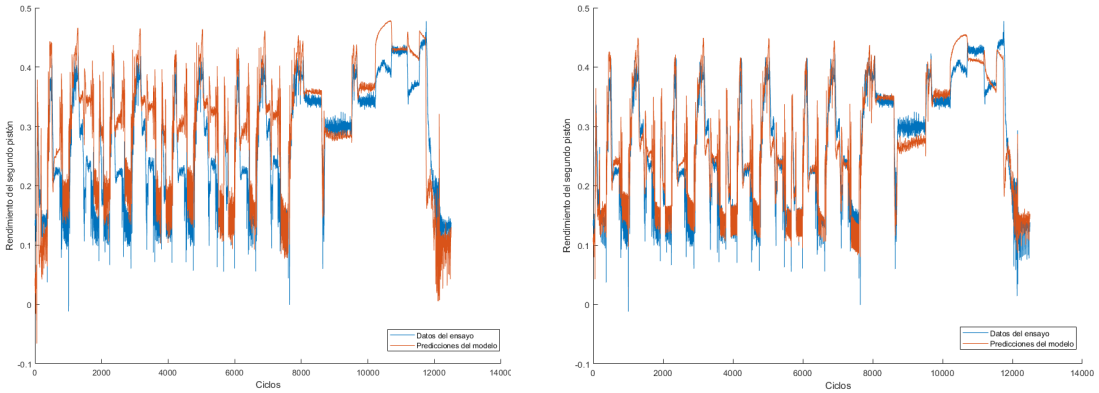
Para la adaptación de este modelo se ha realizado lo propuesto en el inicio de esta sección, preentrenando un modelo y posteriormente adaptándolo a los nuevos datos. Para evaluar las diferencias y el proceso de adaptación se han calculado los errores del modelo para los datos de entrenamiento, validación, adaptación y el conjunto total de datos, para ver como ha variado este modelo. Los resultados obtenidos se encuentran en la siguiente tabla:

	Entrenamiento inicial	Adaptación
Error datos entrenamiento	$0,043$	$0,1629$
Error datos validación	$0,0547$	$0,1772$
Error datos adaptación	0,3235	0,1149
Error todos los datos	0,3208	0,1185
Tiempo de entrenamiento (s)	1846	1264

Tabla 4.2: Comparación de resultados entre entrenamiento normal y adaptación del modelo para los datos del motor EGR de baja presión. Fuente: elaboración propia.

En la tabla destacan dos cosas. Por un lado, observando el error de entrenamiento y validación, como el modelo inicial se había entrenado durante 50000 iteraciones presentan un error muy pequeño. En la figura 4.1a se observa, como, para los primeros ciclos, el modelo se aproxima con precisión a los datos del ensayo pero para el resto presenta un error considerable, esto se ve reflejado en la tabla 4.2, donde para los datos de adaptación y el conjunto de datos totales se aprecia un error muy elevado en comparación con los obtenidos mediante los datos de entrenamiento y validación. Esto justifica la utilización de la adaptación de las redes, mejorando conforme se obtienen nuevos datos para tener un mayor control sobre el proceso de combustión.

Por otra parte, analizando el proceso de adaptación, se observa que el error del entrenamiento y validación aumenta de forma considerable en comparación con el modelo entrenado inicialmente, pese a ello, en la figura 4.1b se observa que no existe una diferencia muy significativa entre los datos de ensayos y la predicción del modelo para los primeros datos, que son los de entrenamiento y validación. Dónde se encuentra una mejora considerable es en los errores obtenidos con los datos de adaptación y con el total de los datos, en los cuales se ha reducido un tercio el error original que habíamos obtenido. Esta mejora sustancial en el error se puede apreciar en la figura 4.1b, donde se aprecia que la predicción del modelo se acerca con precisión a los datos de ensayo, sobre todo durante los primeros 9000 ciclos. Durante la parte final del ensayo no se obtiene tanta precisión, esto es debido a que las primeras tres cuartas partes del ensayo se tratan de ciclos realizados en modo urbano y el último cuarto un ciclo extraurbano, por lo que el modelo a aprendido a modelizar con mucha mayor precisión los ciclos urbanos; esto podría mejorarse con la presencia de un mayor número de datos para los ciclos extraurbanos. Finalmente, cabe destacar que pese a que las figuras 4.1 son tan solo del rendimiento para el segundo cilindro, los resultados son extrapolables al resto de variables, pues carece de sentido llenar el trabajo de gráficas que no aporten nada nuevo.



(a) Rendimiento del segundo cilindro comparado con el rendimiento real del modelo sin adaptar.

(b) Rendimiento del segundo cilindro comparado con el rendimiento real del modelo adaptado.

Figura 4.1: Comparación entre el rendimiento del modelo adaptado y del modelo sin adaptar. Fuente: elaboración propia.

4.2 Extreme Learning Machine

Así como para el entrenamiento de las redes neuronales entrenadas mediante el descenso del gradiente tienen una forma de entrenamiento definida cuando se tienen nuevos datos de los que debe aprender el modelo y adaptarse, para el ELM no se ha desarrollado esto. Para poder llevar a cabo esta labor, en este trabajo se propone que para realizar este proceso de adaptación continua se mantenga lo que se ha estudiado hasta ahora de ELM pero contando con un buffer de datos, de un tamaño definido en el cual se vayan añadiendo los nuevos datos y en el caso de que esté lleno se extraigan los datos más viejos. De esta forma el algoritmo podrá predecir con mayor precisión el ciclo siguiente al contar con el ciclo anterior entre sus datos de entrenamiento. Además se propone también que el periodo de actualización de este buffer sea de un conjunto de datos, es decir que el modelo se adapte cuando tengamos, por ejemplo, 10 datos nuevos. Los parámetros del tamaño de buffer y su periodo de actualización son los que debemos definir con el fin de reducir al máximo el error, sin perder de vista el tiempo de entrenamiento o la capacidad de almacenamiento que debe mantener el computador para un tamaño de buffer excesivamente grande.

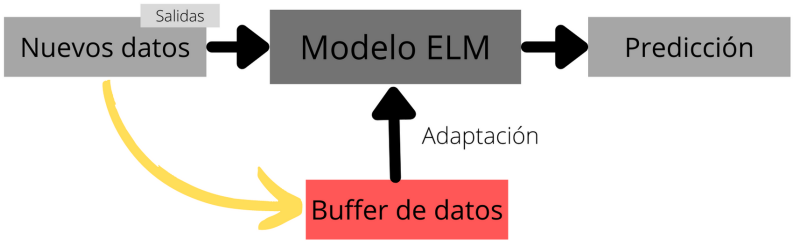


Figura 4.2: Esquema del proceso de adaptación de Extreme Learning Machine. Fuente: elaboración propia.

4.2.1 Motor TJI

Como se ha comentado al inicio de este capítulo, el entrenamiento mediante ELM consiste en la presencia de un buffer de datos donde se irán introduciendo nuevos datos. Con el fin de definir este buffer, es necesario declarar dos variables nuevas, la primera de ellas es el tamaño máximo del buffer (buffer_{max}) y la segunda será el periodo de actualización del buffer ($\text{buffer}_{actualizacion}$), este periodo de actualización se refiere al número de datos nuevos que se tienen que tener para actualizar el buffer y entrenar la red de nuevo. Para definir estos parámetros se van a realizar unas pruebas en las que se calculará el error inicial para los datos de entrenamiento, validación, adaptación y el total de estos; y posteriormente con el modelo final. Además, como el resultado va a depender de los datos que se encuentren en el buffer, y el ELM no precisa de una gran capacidad de cómputo, para cada dato nuevo se calculará también el error con el modelo que se encuentre en ese momento, de esta forma se podrá ver realmente el funcionamiento real del proceso de adaptación.

Dado que para ejecutar el entrenamiento del ELM no se necesita una gran potencia de cálculo, el mayor inconveniente para la implementación real del algoritmo de adaptación del ELM será la capacidad necesaria para almacenar el buffer, el cual nos interesa que sea lo mayor posible, para tener un mayor número de datos y ser capaz de predecir con mayor precisión los siguientes; por otra parte, nos interesa que la actualización del buffer sea lo más rápida posible. Pese a que esto solo son hipótesis teóricas, para la realización de las pruebas se han utilizado dos tamaños de buffer, de 500 y 1000 datos cada uno, y para cada uno de estos se ha variado el periodo de actualización de los datos entre 10, 20, 30 y 40. En todos ellos se han validado los datos con un periodo de diez datos, es decir que se ha realizado la validación con los diez datos siguientes, por lo que para poder comparar estos datos se comparará la media de la precisión, definida en el apartado 4.1.1. Los resultados obtenidos han sido los siguientes:

Tamaño de buffer	Periodo actualización buffer	Precisión media (%)	Tiempo de entrenamiento (s)
500	10	85,62	604,85
500	20	85,13	597,19
500	30	84,73	531,73
500	40	84,52	498,36
1000	10	85,60	482,05
1000	20	85,13	482,55
1000	30	84,74	494,42
1000	40	84,44	520,40

Tabla 4.3: Resultados de adaptación mediante ELM para distintos tamaños de buffer y distintos periodos de actualización para el motor TJI. Fuente: elaboración propia.

Como se puede observar en la tabla 4.3, no existe una gran diferencia en los resultados para el proceso de clasificación del missfire. Como se ha comentado al inicio de esta sección, un mayor periodo de actualización nos aportaría datos más precisos, pero tampoco hay una diferencia muy significativa ente ellos. Por el contrario, en la hipótesis inicial se planteaba que un mayor tamaño de buffer daría mejores resultados, pero no se aprecia una gran diferencia entre un tamaño de buffer y otro, esto es posible que sea debido a que un mayor número de datos induce error en el modelo debido a que los datos más antiguos no tendrán un comportamiento similar a los más

nuevos, si solo se tuvieran los datos más nuevos este problema podría verse reducido. Pese a ello, como ya se ha dicho anteriormente, no existe una gran diferencia de precisión entre uno y otro. Dónde si que aparece una mayor diferencia es en el tiempo de entrenamiento, dónde se aprecia que para los periodos de actualización más pequeños existe una diferencia de casi 100 segundos para el entrenamiento de ELM con distintos tamaño de buffer. Dado que la diferencia en la precisión es despreciable se tomará como prueba con mayor precisión aquella que ha requerido un menor tiempo de ejecución, es decir un tamaño de buffer de 1000 datos y un periodo de actualización cada 10 datos.

4.2.2 Motor diésel con EGR de baja presión

Al igual que en el motor anterior, para determinar el tamaño de buffer y el periodo de actualización de este se van a realizar unas pruebas con el fin de determinar el tamaño de este. Tras los resultados obtenidos del ensayo anterior, se puede prever que estos parámetros no van a afectar en gran medida al error de nuestro modelo, pese a ello, al ser un problema de regresión y no de clasificación, se va a tratar de la misma manera.

Esta vez, para el análisis de los resultados se ha calculado el error de la red, y el error porcentual medio por pistón de cada una de las salidas, que recordemos son CA10, CA50, CA90 y rendimiento de cada pistón. También se han medido los tiempos de entrenamiento para observar su comportamiento y compararlo más adelante, obteniendo los siguientes resultados:

Tamaño de buffer	500	500	500	500	1000	1000	1000	1000
Periodo de actualización	10	20	30	40	10	20	30	40
Error medio	0,051	0,0534	0,0565	0,0583	0, 051	0, 0526	0, 0537	0, 0556
Error medio porcentual CA10 (%)	23, 993	24, 8763	26, 2398	27, 2076	24, 2649	24, 6475	24, 1023	26, 0944
Error medio porcentual CA50 (%)	14, 0064	14, 435	14, 9882	15, 3682	14, 0434	14, 3078	14, 5294	14, 9919
Error medio porcentual CA90 (%)	9, 9882	10, 1768	10, 6556	10, 7907	10, 0224	10, 1196	10, 3099	10, 5099
Error medio porcenutal rend (%)	5, 9364	6, 3561	6, 6314	6, 8936	5, 9366	6, 211	6, 4021	6, 6792
Tiempo de ejecución (s)	45, 1406	41, 5308	38, 7886	37, 335	45, 736	40, 461	39, 303	38, 447

Tabla 4.4: Resultados de adaptación mediante ELM para distintos tamaños de buffer y distintos periodos de actualización para el motor con EGR de baja presión. Fuente: elaboración propia.

Al igual que ocurría en el motor TJI, la diferencia entre distintos tamaños de buffer es irrelevante, al igual que ocurre con la variación del periodo de actualización, obteniéndose un error algo más pequeño en los periodos de actualización más pequeños. En este apartado, lo que se observa claramente es que a mayor periodo de actualización menor tiempo de ejecución, esto tiene más sentido que en el apartado anterior, dónde no se apreciaba esta relación. Esto es debido a que para un mayor periodo de actualización será necesario acceder al buffer más veces, lo cual implica un retraso para realizar el entrenamiento. A la vista de los resultados, y viendo que el tiempo de ejecución es despreciable, se puede afirmar que las opciones con menor periodo de actualización son las que nos confieren mejores resultados.

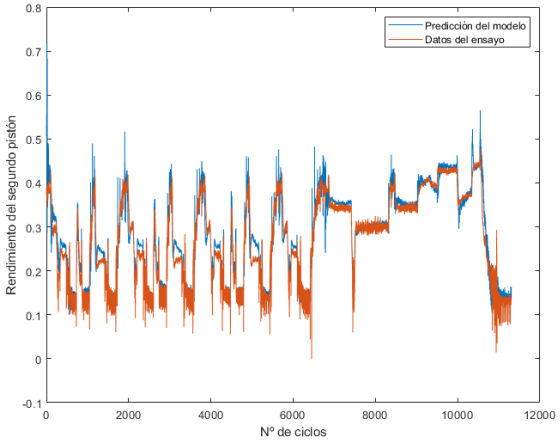


Figura 4.3: Rendimiento del segundo cilindro comparado con los datos obtenidos del ensayo tras el proceso de adaptación mediante ELM para un tamaño de buffer y periodo de actualización de 500 y 10 respectivamente. Fuente: elaboración propia.

Por otra parte, en la figura 4.3, se observa como el modelo de adaptación se comporta de manera muy estable en todo el rango de datos, no solo en los ciclos urbanos, que son muy similares.

Dado que los modelos basados en ELM utilizan un buffer de datos de distinto tamaño, para este motor se va a plantear que ocurre si ocurren errores en la medición de los datos para observar como afectan a la predicción de los futuros datos, para ello, con un tamaño de buffer de 500 datos y un periodo de actualización de 10 datos, se va a proceder a insertar en el 5 % de los datos de adaptación un error cuyo valor de entrada a la red es 1, una vez normalizado. Con estos datos erróneos se ha obtenido lo siguiente:

Error medio	$0,055$
Error medio porcentual CA10	$25,1846$
Error medio porcentual CA50	$14,2678$
Error medio porcentual CA90	$10,3693$
Error medio porcentual rend	$6,4179$
Tiempo de ejecución (s)	$44,1298$

Tabla 4.5: Resultados tras insertar errores en ELM

A la vista de estos resultados, y apoyándonos en la figura 4.4, se observa que el error no ha aumentado sustancialmente, ya que se inserta error en muy pocos datos. A pesar de ello, se observa en la figura 4.4 como existen una serie de picos en la predicción del modelo que, aun que siguen la tendencia de la salida, se desmarcan considerablemente de los datos ensayados, sobre todo en el ciclo extraurbano. Con el fin de corregir esto se podría aplicar algún tipo de filtro a la entrada previo que sea capaz de detectar si esa entrada tiene sentido físico o se ha producido un error en el sistema electrónico de medida.

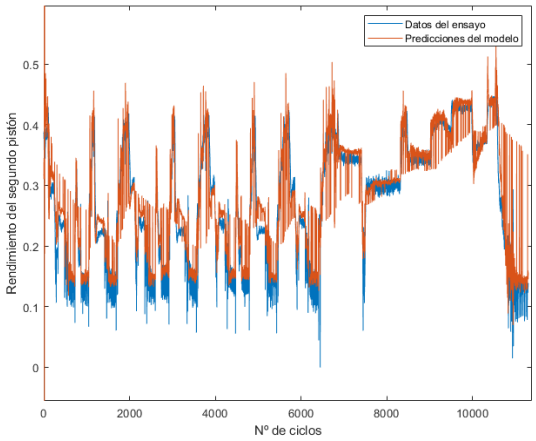


Figura 4.4: Rendimiento del segundo cilindro comparado con los datos obtenidos del ensayo tras el proceso de adaptación mediante ELM con datos erróneos. Fuente: elaboración propia.

Capítulo 5

Resultados

En este apartado se va a justificar la selección del modelo más adecuado para cada motor dadas las pruebas y el estudio realizado anteriormente.

A lo largo de los dos apartados anteriores se ha realizado un estudio acerca del entrenamiento y el proceso de adaptación de los modelos. A grosso modo, hemos observado que únicamente, para los datos de entrenamiento y validación los modelos de redes neuronales basados en el entrenamiento mediante el descenso del gradiente se comportan mucho mejor, pero requieren de un tiempo mucho mayor para su entrenamiento. En cambio, para el proceso de adaptación, el Extreme Learning Machine se comporta de una manera excepcional. Para comprobar esto, a partir de todas las pruebas y el proceso de adaptación realizado se va a proceder a definir cual es el mejor modelo para cada motor.

Antes de continuar, cabe aclarar que todas las pruebas realizadas han sido ejecutadas en un PC con las siguientes características:

- Placa base: MSI B450 Gamin Plus
- Procesador: AMD Ryzen 5 2600 3.4 GHz
- Memoria RAM: 2x8GB DDR4
- Tarjeta gráfica: GeForce GTX 1660

Por lo que es posible que al tratar de recrearlas en otro ordenador, se obtengan tiempos ejecución distintos.

5.1 Motor TJI

Recordemos que el estudio de nuestro motor TJI es un problema de clasificación, en el que pretendemos predecir si se produce o no combustión dentro del cilindro. Para ello nuestro modelo calcula la probabilidad de que se produzca combustión, y si es menor del 10 % se considera que no se producirá. Como se ha comentado a lo largo de todo el estudio, una buena forma de comparar distintos modelos es mediante el número de aciertos o la precisión de aciertos, que es una forma de normalizar estos datos. Como el objetivo final de este trabajo está más enfocado al

proceso de adaptación para unas redes pre-entrenadas, nos centraremos en los resultados de los modelos estudiados para este motor tras el proceso de adaptación. Estos resultados se recogen en la siguiente tabla:

	Redes Neuronales	ELM
Error entrenamiento	0,3111	0,3044
Error medio tras adaptación	0,4139	0,3721
Precisión media	79,22	85,60
Tiempo de entrenamiento (s)	5541	482,05

Tabla 5.1: Comparación de redes neuronales y ELM. Fuente: elaboración propia.

A la vista de los resultados queda claro que el modelo de ELM obtiene una precisión de un 5 % mayor que el modelo de redes neuronales con entrenamiento basado en el descenso del gradiente. A parte, el tiempo de entrenamiento y validación es aproximadamente 10 veces menos para el ELM por lo que no requiere de una capacidad de cálculo demasiado elevada. Por lo tanto, podemos concluir el estudio de este motor asegurando que el ELM se comporta mejor, requiriendo de un menor tiempo de entrenamiento y por tanto de una menor capacidad de cálculo.

A lo largo del trabajo se ha considerado como acierto la predicción correcta tanto de si se produce combustión como si no, pero con el fin de estudiar mejor los resultados, se va a desgranar esto, de tal forma que tendremos dos tipos de errores y dos tipos de aciertos:

- Acierto tipo 1: que se produzca combustión y que el modelo prediga que se va a producir combustión.
- Acierto tipo 2: que no se produzca combustión y que el modelo prediga que no se va a predecir combustión.
- Error tipo 1: que se produzca combustión y que el modelo prediga que no se produce combustión.
- Error tipo 2: que no se produzca combustión y que el modelo prediga que se produce combustión.

De las redes neuronales entrenadas mediante el descenso del gradiente se han obtenido los siguientes errores:

		Salida del modelo	
		Combustión	No combustión
Datos de ensayo	Combustión	Aciertos Tipo 1 38,93	Errores Tipo 1 19,1
	No combustión	Errores Tipo 2 1,67	Aciertos Tipo 2 40,28

Tabla 5.2: Separación de precisión para la red entrenada mediante el descenso del gradiente con un umbral de 0,1.

Y del Extreme Learning Machine:

		Salida del modelo	
		Combustión	No combustión
Datos de ensayo	Combustión	Aciertos Tipo 1 81,1	Errores Tipo 1 1,937
	No combustión	Errores Tipo 2 12,463	Aciertos Tipo 2 4,49

Tabla 5.3: Separación de precisión para la red entrenada mediante ELM con un umbral de 0,1.

De estas tablas se pueden extraer un par de conclusiones, y es que al principio del apartado 3.1.2 se fijó el umbral que definía si en función de la salida del modelo se predecía si había combustión o no en el motor TJI; este umbral se fijó en 0.1, y es determinante para calcular la precisión de la red. Como se observa en la tabla 5.2, el mayor tipo de errores se producen cuando si que ocurre combustión pero el modelo predice que no, esto podría ser resuelto variando este umbral hasta encontrar el que nos maximice el porcentaje total de aciertos. Lo mismo ocurre en la tabla 5.3, dónde apenas se producen aciertos del tipo 2, ya que cuando no se produce combustión, el modelo considera que sí. Para ver como afecta este umbral al error, se muestran a continuación para la misma salida del modelo pero esta vez con un umbral de 0,5, primero para las redes neuronales entrenadas mediante el descenso del gradiente:

		Salida del modelo	
		Combustión	No combustión
Datos de ensayo	Combustión	Aciertos Tipo 1 38.801	Errores Tipo 1 19.1
	No combustión	Errores Tipo 2 1,67	Aciertos Tipo 2 40,418

Tabla 5.4: Separación de precisión para la red entrenada mediante el descenso del gradiente con un umbral de 0,5.

Y del Extreme Learning Machine:

		Salida del modelo	
		Combustión	No combustión
Datos de ensayo	Combustión	Aciertos Tipo 1 75,05	Errores Tipo 1 7,43
	No combustión	Errores Tipo 2 6,81	Aciertos Tipo 2 10,71

Tabla 5.5: Separación de precisión para la red entrenada mediante ELM con un umbral de 0,5.

Tras variar este umbral, no se observa una gran diferencia en la salida del modelo, de echo el porcentaje total de aciertos y errores se mantiene más o menos constante pese a variar este umbral, lo que varía algo más es la distribución de los aciertos y errores en sus tipos. Esto se aprecia más en el ELM, donde tras variar el umbral se distribuye de manera más uniforme el porcentaje de errores.

5.2 Motor diésel con EGR de baja presión

Para el estudio de este motor, se ha tratado de predecir una serie de parámetros característicos de la combustión, estos parámetros solo pueden ser medidos en un banco de ensayos, por lo que es de gran interés predecirlos durante su funcionamiento, para optimizar el funcionamiento del motor y ser capaces de maximizar su rendimiento, consiguiendo extraer el máximo de energía posible del proceso de combustión y reduciendo el consumo de combustible. Para decidir que modelo se adapta mejor se va a comparar el error medio del modelo y el tiempo de ejecución del entrenamiento y adaptación, esto se hará para el mejor modelo obtenido de redes neuronales y ELM. Esto se recoge en la siguiente tabla:

	ELM	Backpropagation
Error medio entrenamiento	0,0516	0,043
Error medio tras adaptación	0, 051	0, 1185
Error medio porcentual CA10	23, 993	25, 819
Error medio porcentual CA50	14, 0064	19, 941
Error medio porcentual CA90	9, 9882	20, 9185
Error medio porcentual rend	5, 9364	5, 9487
Tiempo de ejecución (s)	45, 1406	3110

Tabla 5.6: Comparación de redes neuronales y ELM. Fuente: elaboración propia.

Observando los resultados anteriores, se puede ver claramente que el Extreme Learning Machine obtiene la mitad de error que las redes neuronales con entrenamiento basado en el descenso del gradiente. A parte, el tiempo de entrenamiento es mucho menor para el ELM, lo que facilita el proceso de adaptación en el propio controlador del motor, al requerir de una capacidad de cómputo mucho menor para el proceso de entrenamiento. En cuanto a los errores en cada salida, se puede observar que a excepción del rendimiento, en todos existe una diferencia notable entre un método de entrenamiento y otro, como se puede observar en la figure 5.1

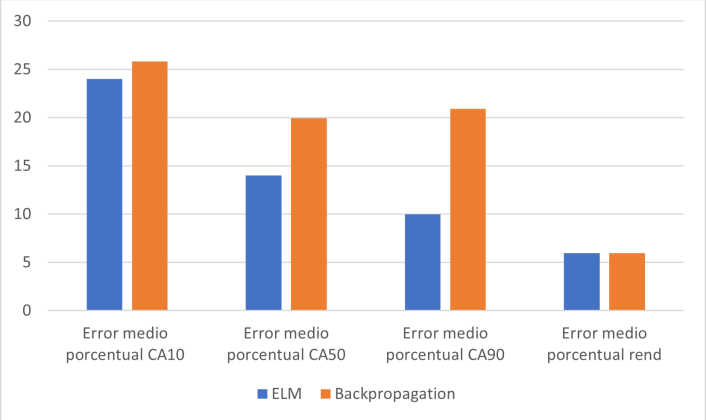


Figura 5.1: Comparación entre errores medios porcentuales para cada salida de cada modelo. Fuente: elaboración propia.

Por otra parte, en las figura 5.3, se puede observar esta diferencia de error, ajustando el modelo mejor tras el proceso de adaptación ELM, sobre todo esta diferencia se puede apreciar en el ciclo extraurbano, donde las redes neuronales no acaban de ajustarse adecuadamente a los datos obtenidos del ensayo. Por lo que se puede concluir que en este caso, el ELM tras el proceso de adaptación se ajusta mucho mejor a los datos obtenidos en el ensayo.

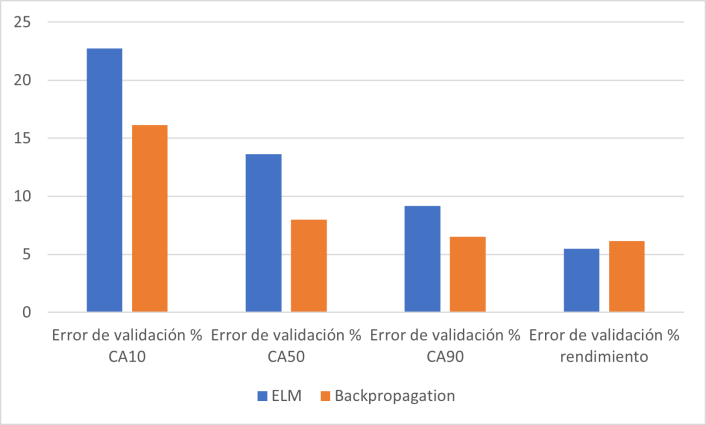
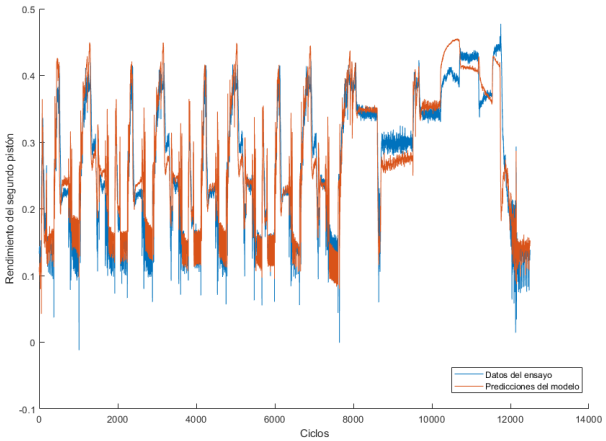
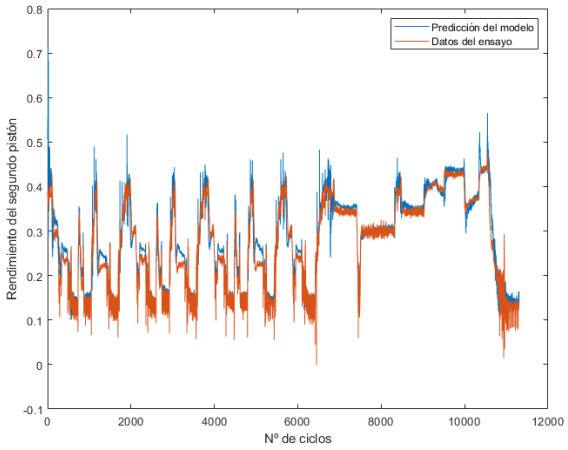


Figura 5.2: Comparación entre errores medios porcentuales para cada salida de cada modelo para los datos de entrenamiento. Fuente: elaboración propia.

De la tabla 5.6 destaca algo que se vio durante el capítulo 3, dónde el error de entrenamiento y validación para redes neuronales entrenadas mediante el descenso del gradiente, era por lo general más pequeño, por lo que si se dispone del tiempo y la capacidad de cálculo necesaria para el entrenamiento de este tipo de modelo el error de adaptación se podría reducir considerablemente; que, como se observa en la figura 5.2 solo con el proceso de entrenamiento los modelos basados en redes neuronales entrenadas con el descenso del gradiente obtienen un error considerablemente menor.



(a) Rendimiento del segundo pistón tras el proceso de adaptación mediante redes neuronales para la totalidad de los datos.



(b) Rendimiento del segundo pistón tras el proceso de adaptación mediante ELM para los datos de adaptación.

Figura 5.3: Comparación de rendimiento del segundo pistón para distintos modelos. Fuente: elaboración propia.

Capítulo 6

Conclusiones

Tras todo el desarrollo del proyecto, se muestran las conclusiones obtenidas tras el estudio de los resultados y el planteamiento inicial de los objetivos.

- A priori, únicamente para los datos de entrenamiento y los datos de validación las redes neuronales presentan un menor error y una mayor robustez frente al Extreme Learning Machine, por el contrario, como ya se conocía, el Extreme Learning Machine muestra un tiempo de entrenamiento infinitamente menor que el entrenamiento mediante el descenso del gradiente, debido a que no es necesaria la iteración sobre los datos para calcular la matriz de pesos.
- Por otra parte, tras el proceso de adaptación el Extreme Learning Machine ha obtenido datos muchos mejores, llegando incluso a obtener la mitad del error en el motor EGR de baja presión, por lo que el ELM se plantea como un modelo muy interesante cuando es necesaria la adaptación de los datos, sin embargo cuando se producen errores en la medida, estos parecen afectar en buena parte por lo que sería interesante implementar un sistema de control de errores.
- De los dos puntos anteriores, se puede concluir que en función de la velocidad de envejecimiento de nuestro motor sería interesante aplicar un tipo de modelo u otro. Si estamos trabajando con un motor de lento envejecimiento sería interesante la aplicación de los modelos de redes neuronales basados en el descenso del gradiente, puesto que, como se ha comentado a lo largo del trabajo, se puede reducir su error considerablemente si se dispone del tiempo y la capacidad de cálculo suficiente para el proceso tanto de entrenamiento como de validación. Por el contrario, si el motor es de envejecimiento rápido, sería más interesante un modelo que se adapte a una velocidad adecuada, de ahí el uso del ELM dada su escasa necesidad de capacidad de cálculo.
- No se puede dejar pasar lo último visto en el apartado 5.1, dónde en función del umbral definido se obtiene un tipo de errores u otro, variando considerablemente la precisión del modelo. Este es un parámetro que no puede quedar tan al aire de cara a trabajos futuros y una implementación real del modelo.

- Finalmente, tanto en el motor TJI, como en el motor EGR de baja presión se ha obtenido una gran precisión, por lo general mayor del 80 %, lo cual son unos rangos de valores más que aceptables para su implementación y su uso en los nuevos MCIA.

De esta forma, se dan por cumplidos los objetivos, tras haber sido capaces de desarrollar y estudiar distintos modelos de redes neuronales para después adaptarlos a nuevos ciclos. También se reitera lo comentado en el tercer punto de este capítulo, dónde a la hora de la implementación definiremos el tipo de modelo en función de la velocidad de envejecimiento del motor que se esté utilizando.

6.1 Trabajos futuros

Hasta el momento, solo se han diseñado los modelos para los datos obtenidos de distintos ensayos de este motor, en un futuro, sería muy interesante obtener muchos más datos iniciales para tener un mapa completo del funcionamiento diario del motor, de tal forma que la fase de pre-entrenamiento sería mucho más precisa.

Por otra parte, también sería de interés observar el proceso de adaptación del modelo a la vez que el motor se encuentra funcionando, esto implicaría la implantación del modelo en la unidad de control del motor con el desarrollo que esto conlleva.

Otra línea futura sería la implementación en un bucle de control de algunas salidas de los modelos, de esta forma se podría controlar, por ejemplo, el proceso de combustión de una manera mucho más precisa, y se podría optimizar adecuadamente el proceso de combustión.

Un filtrado de errores, como se ha comentado anteriormente, para el Extreme Learning Machine, nos permitiría aumentar su robustez frente a errores, haciendo más fiable la red frente a errores de la instrumentación.

Finalmente, si se obtuvieran datos de emisiones de gases, se podría tratar de predecir estos gases, y mediante un bucle de control, como el que se ha comentado anteriormente, se podrían reducir estas emisiones, siguiendo con la tendencia de reducción de gases que tan necesaria es en estos momentos.

Parte II

Presupuesto

En este apartado se va a detallar el desarrollo del presupuesto, una valoración económica del proyecto "Desarrollo de un modelo basado en redes neuronales adaptativas aplicadas para optimizar el funcionamiento de la nueva generación de MCIA", acorde al documento *Recomendaciones en la elaboración de presupuestos en actividades de I+D+I, Revisión 2018"*. de la UPV.

- CUADRO Nº1: PRECIOS DE LA MANO DE OBRA.

Código	Descripción de la mano de obra	Precio (€/h)
MO1	Titular Universitario	39,00
MO2	Graduado en Ingeniería en Tecnologías Industriales	20,00
MO3	Investigador predoctoral	25,00

- CUADRO Nº2: PRECIOS DE LOS MATERIALES PUESTOS EN OBRA.

Código	Descripción del material	Precio (€)	Periodo de amortización (años)	Precio (€/h)
MA1	Ordenador	868,71	6	12,07
MA2	Licencia MatLab	2000	6	27,78

- CUADRO Nº3: PRECIOS DESCOMPUESTOS.

Nº de orden	Descripción de las unidades de obra	Unidad	Rendimiento	Precio (€)	Importe (€)
UO1	Desarrollo de los modelos de redes neuronales y Extreme Learning Machine	h			73,05
MO1	Titular Universitario	h	0,05	39	1,95
MO2	Graduado en Ingeniería de Tecnologías Industriales	h	1	20	20,00
MO3	Investigador predoctoral	h	0,02	25	0,5
MA1	Ordenador	h	1	12,07	22,22
MA2	Licencia MatLab	h	0,8	27,78	1,70
	Costes directos complementarios		0,03	56,74	1,70
	Costes directos				58,44
	Costes indirectos		0,25	58,44	14,61

Nº de orden	Descripción de las unidades de obra	Unidad	Rendimiento	Precio (€)	Importe (€)
UO2	Testeo de los modelos de redes neuronales y Extreme Learning Machine	h			77,55
MO1	Titular Universitario	h	0,01	39	1,95
MO2	Graduado en Ingeniería de Tecnologías Industriales	h	1	20	20,00
MA1	Ordenador	h	1	12,07	22,22
MA2	Licencia MatLab	h	1	27,78	27,78
	Costes directos complementarios		0,03	60,23	1,81
	Costes directos				62,04
	Costes indirectos		0,25	62,04	15,51

Nº de orden	Descripción de las unidades de obra	Unidad	Rendimiento	Precio (€)	Importe (€)
UO3	Redacción de la documentación	h			43,80
MO1	Titular Universitario	h	0,05	39	1,95
MO2	Graduado en Ingeniería de Tecnologías Industriales	h	1	20	20,005
MA1	Ordenador	h	1	12,07	22,22
	Costes directos complementarios		0,03	34,02	1,02
	Costes directos				35,04
	Costes indirectos		0,25	35,04	8,76

- CUADRO Nº4: PRECIOS UNITARIOS

Nº de orden	Descripción de la unidad de obra	Unidad	Precio (€)
UO1	Desarrollo de los modelos de redes neuronales y Extreme Learning Machine	h	73,05
UO2	Testeo de los modelos de redes neuronales y Extreme Learning Machine	h	77,55
UO3	Redacción de la documentación	h	43,80

- CUADRO Nº5: ESTADO DE MEDICIONES

Nº de orden	Descripción de la unidad de obra	Unidad	Medición
UO1	Desarrollo de los modelos de redes neuronales y Extreme Learning Machine	h	130,00
UO2	Testeo de los modelos de redes neuronales y Extreme Learning Machine	h	150,00
UO3	Redacción de la documentación	h	50,00

- CUADRO Nº5: ESTADO DE MEDICIONES

Nº de orden	Descripción de la unidad de obra	Unidad	Precio (€)	Medición	Importe (€)
UO1	Desarrollo de los modelos de redes neuronales y Extreme Leaning Machine	h	73,05	130,00	9496,46
UO2	Testeo de los modelos de redes neuronales y Extreme Learning Machine	h	77,55	150,00	11632,54
UO3	Redacción de documentación	h	43,80	50,00	2190,04
	Total Presupuesto de Ejecución Material (PEM)				23319,04
	Gastos Generales (12 % del PEM)				2798,28
	Beneficio Industrial (6 % del PEM)				1399,14
	Total Presupuesto de Ejecución por Contrata (PEC)				27516,46
	IVA (21 % del PEC)				5778,46
	PRESUPUESTO BASE DE LICITACIÓN				33294,92

El coste total del proyecto asciende a **TREINTA Y TRES MIL DOSCIENTOS NOVENTA Y CUATRO EUROS Y NOVENTA Y DOS CÉNTIMOS**.

Bibliografía

B.Pla, la Morena, J. D. & Jiménez, P. B. I. (2020). Cycle-to-cycle combustion variability modelling in SI engines for control purposes. *International Journal of Engine Research* (vid. pág. 40).

Cruz, J. R. S. y col. (2002). *Procesos y tecnología de máquinas y motores térmicos*. Editorial Universitat Politècnica de València. (Vid. págs. 4, 32).

Hastie, T., Tibshirani, R. & Friedman, J. (2001). *The Elements of Statistical Learning*. Springer-Verlag. (Vid. pág. 9).

Hornik, K., Tinchcombe, M. & White, H. (1989). Multilayer Feedforward Networks are Universal Approximators. *Pergamon Press plc.* (vid. pág. 16).

Huang, G.-B. y col. (2005a). On-Line Sequential Extreme Learning Machine. *Nanyang Technological University* (vid. pág. 30).

Huang, G.-B., Zhu, Q.-Y. & Siew, C.-K. (2005b). Extreme learning machine: Theory and applications. *ScienceDirect* (vid. págs. 28-30).

Klimstra, J. (1985). The Optimum Combustion Phasing Angle—A Convenient Engine Tuning Criterion. *SAE Technical Paper* (vid. pág. 40).

Kumar, S. K. (2017). On weight initialization in deep neural networks. *Cornell University* (vid. pág. 26).

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, Volume 5* (vid. pág. 13).

A proposal for the Dartmouth Summer Research Project on Artificial Intelligence. (1955). (Vid. pág. 8).

Sutton, R. S. & Barto, A. G. (2014). *Reinforcement Learning: An Introduction*. The MIT Press. (Vid. pág. 10).

Toulson, E., Schock, H. J. & Attard, W. P. (2010). A Review of Pre-Chamber Initiated Jet Ignition Combustion Systems. *SAE Technical Paper* (vid. pág. 6).