UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DESARROLLO DE UN MODELO BASADO EN REDES NEURONALES ARTIFICIALES PARA LA PREDICCIÓN DE EMISIONES CONTAMINANTES EN UN MOTOR DIESEL DI

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Mecánica

Junio del 2019

AUTOR:

Checa Martínez, Adrián

TUTOR:

López Sánchez, José Javier

motores térmicos

AGRADECIMIENTOS

Me gustaría agradecer a Javi, mi tutor de proyecto, que ha estado siempre disponible cuando lo he necesitado y ha solucionado todos los problemas y dudas que he tenido.

Especial agradecimiento también a mi familia y amigos por aguantarme y acompañarme por el duro camino que he recorrido hasta la realización de este proyecto.

ÍNDICE

1.-INTRODUCCIÓN

1.1.- Contexto de la investigación

Las emisiones contaminantes son un problema a nivel global, siendo su regulación cada vez más estricta y necesaria para paliar consecuencias como el calentamiento global, el empeoramiento de la calidad del aire y la lluvia ácida entre otras. Estas restricciones han hecho que a lo largo de los años el diseño de los motores esté enfocado a minimizar dichas emisiones de manera progresiva, pero la formación de los diferentes contaminantes durante la combustión es un proceso muy complejo que depende tanto de condiciones locales como de la configuración del motor. Llegar a una solución de compromiso entre optimizar la economía del combustible y cumplir con las regulaciones actuales de emisiones es un trabajo de gran dificultad.

El sector de la industria dedicado a esto ha experimentado un gran desarrollo tecnológico recientemente, creando nuevas estrategias como el control electrónico, sistemas de inyección múltiple y métodos de recirculación de los gases de escape (EGR). A su vez, los ensayos de ciclos en MCIA son cada vez más precisos, y cubren un mayor rango de condiciones de funcionamiento. Todo esto hace de la creación de modelos y herramientas de predicción para el diseño y calibrado de estos motores un campo científico de interés.

Uno de estos modelos es el VEMOD, desarrollado como una herramienta autónoma para simular ensayos de ciclos estándar por CMT – motores térmicos. Para la predicción de los resultados este utiliza un modelo virtual de un motor con varios submodelos encargados cada uno de un aspecto del modelado, como el sistema de lubricación, sobrealimentación, refrigeración, o combustión y emisiones.

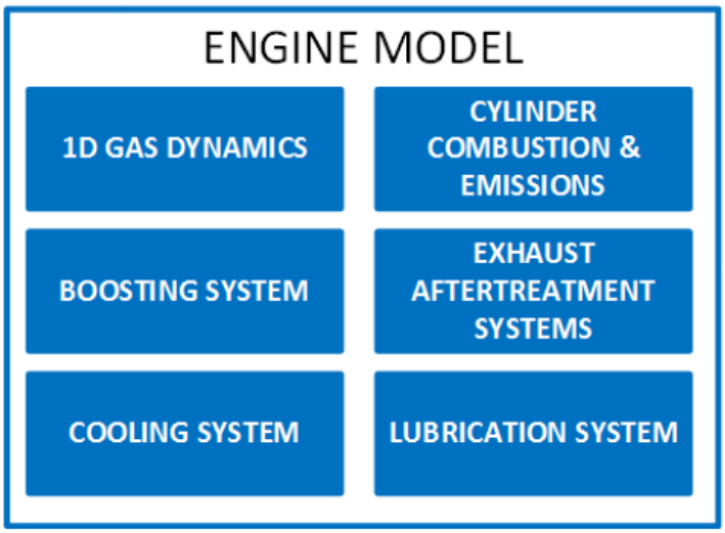

Fig. 1.- Submodelos que engloba el modelo del motor

Centrándose en este último, un submodelo se combina con el proceso de combustión para predecir las emisiones de NOx, CO, HC y partículas, en función de las condiciones de operación del motor. La obtención de los NOx se puede llevar a cabo mediante un modelo físico-térmico, sin embargo, el resto de las emisiones se ven afectadas por demasiados fenómenos locales, resultando imposible el uso de ningún modelo físico.

Hasta ahora, el procedimiento ha sido utilizar correlaciones empíricas para cada una de las emisiones. Estas correlaciones, como por ejemplo regresiones lineales o no lineales, no han dado muy buenos resultados. Comienza así una investigación de nuevas formas más fiables para la predicción de estos contaminantes.

1.2.- Motivo del proyecto y estado de la cuestión

Las redes neuronales artificiales (ANNs) son una herramienta potente que ha demostrado ser de gran utilidad a la hora de resolver diversos problemas que con otros métodos no se ha logrado. Algunas ANN ya han sido desarrolladas y aplicadas en la industria, incluyendo el ámbito del modelado de motores.

Gracias a uno de los submodelos del VEMOD y estudio de CFD para observar el comportamiento en transitorio, se dispone de las condiciones locales junto a la temperatura y fracción másica de las especies químicas en cada instante de la combustión. A raíz de esto se propuso intentar utilizar redes neuronales para intentar mejorar el submodelo de predicciones.

En un trabajo previo en el mismo contexto, que dio lugar a un TFM, Héctor Amino creó una red neuronal en MATLAB para cada una de las emisiones: HC, CO y partículas (SOOT). Las redes de HC y CO consiguieron una precisión notable a la hora de predecir emisiones tanto en condiciones estacionarias como en transitorias. La red de SOOT logró resultados aceptables en estacionario, sin embargo, para casos en transitorio poseía un error inaceptable. Más tarde se comprobó que ante una pequeña perturbación en los ensayos, y por lo tanto en los datos de entrada de la red, se reducía drásticamente su fiabilidad.

1.3.- Propuesta de investigación y objetivo del proyecto

Conocido el estado actual de la primera aproximación a las ANN, se propone mejorar el conjunto de redes del que se dispone. Un método con que se pretende realizar la mejora es la adición de datos de nuevos ciclos experimentales en transitorio realizados con el motor a baja y a alta temperatura ambiente. Gracias a los nuevos datos de los transitorios, se prevé que la red tendrá suficiente variedad de información de entrada para una mejor predicción. Además, se tratará de diseñar un método con el que se pueda determinar la estabilidad de la red y analizar su comportamiento.

El objetivo último del proyecto es obtener un conjunto de redes neuronales para las emisiones de HC, CO y SOOT que posean una precisión excelente y a su vez sean robustas. De esta forma, en caso de haber una pequeña variación o perturbación de cualquier tipo en los datos que utiliza para la predicción, éstas serán capaces de dar resultados con la misma fiabilidad.

2.- MARCO TEÓRICO

2.1.1.- Introducción a las redes neuronales

Pese a no existir una definición universal para una red neuronal debido a la gran variedad de campos en los que se puede utilizar, se puede generalizar con una definición para este trabajo. Una red neuronal artificial es un sistema inspirado en el sistema neuronal biológico compuesto por múltiples elementos simples de procesado operando en paralelo. Su función viene dada por la estructura que tenga la red y los diversos parámetros que la componen, como las uniones entre los elementos de procesado, o el proceso que éstos desarrollan. Para poder entender el funcionamiento de la red neuronal se comenzará definiendo el elemento más simple que compone este sistema: una neurona.

Una neurona es la base de una red neuronal. Representa un espacio de memoria donde se almacena información. Habitualmente se comparan estas neuronas con las de los seres humanos, ya que se comportan como una versión simplificada de las que componen nuestro sistema biológico.

Las conexiones neuronales del sistema nervioso que transmiten señales de unas células o neuronas a otras son simuladas por la ANN. En el modelo artificial, la información que recibe o transmite una neurona se transfiere mediante los pesos (weights). Estos elementos del modelo son de vital importancia debido a que contienen el peso o nivel de importancia de la información que se transmite entre las neuronas.

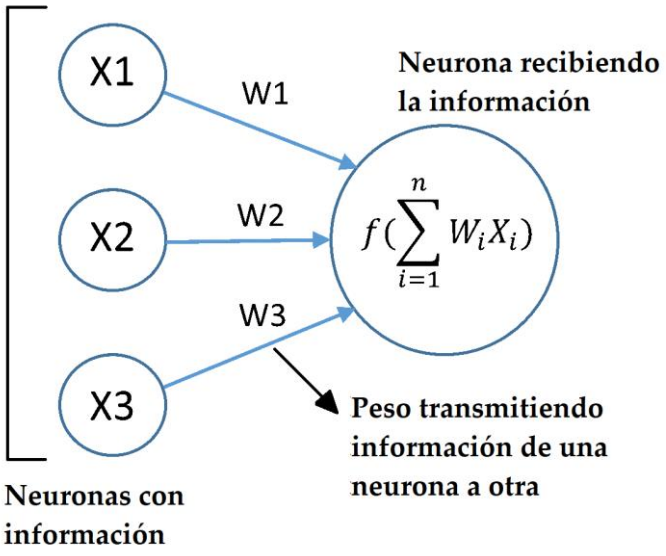

Fig. 2.- Transmisión de información entre neuronas.

2.1.2.- Componentes de una red neuronal y su estructura

Descritos los dos elementos básicos de una red neuronal: las neuronas, también llamadas nodos, y los pesos, se procede a definir el resto de los componentes más complejos que la estructuran.

Las redes neuronales tienen como mínimo dos capas, que son donde se agrupan las neuronas para poder estructurar la red. La primera es la capa de entrada y contiene los parámetros iniciales, es decir, la información que se utilizará para procesar el problema o cuestión a solucionar. La capa de salida es aquella que contiene la solución obtenida, y puede ser o bien un numero o una categoría, dependiendo del tipo de red. Si existe más de una capa de neuronas entre la capa de entrada y la de salida se denomina capa oculta. Los pesos conectan las neuronas de una capa a otra dependiendo de la arquitectura de la red. En el caso desarrollado en este proyecto, una arquitectura básica se ajusta correctamente a la complejidad del problema. Siguiendo esta arquitectura, cada neurona de una capa se conecta con cada neurona de la siguiente.

La adquisición de la información en una neurona viene seguida de su procesado mediante una función de activación. Son básicamente funciones que definen cómo va a ser la salida de información de la neurona. Existen gran variedad de funciones de activación dependiendo de la salida que se requiera que tenga el nodo, como por ejemplo la función escalón, que define una salida binaria (1 o 0), o la función sigmoidea, que acota la salida en valores reales entre cero y uno. Por último, los Bias son elementos que contiene cada neurona y determinan la forma en que la neurona enviará información, desde decidir si envían o no información hasta regular la cantidad de información enviada.

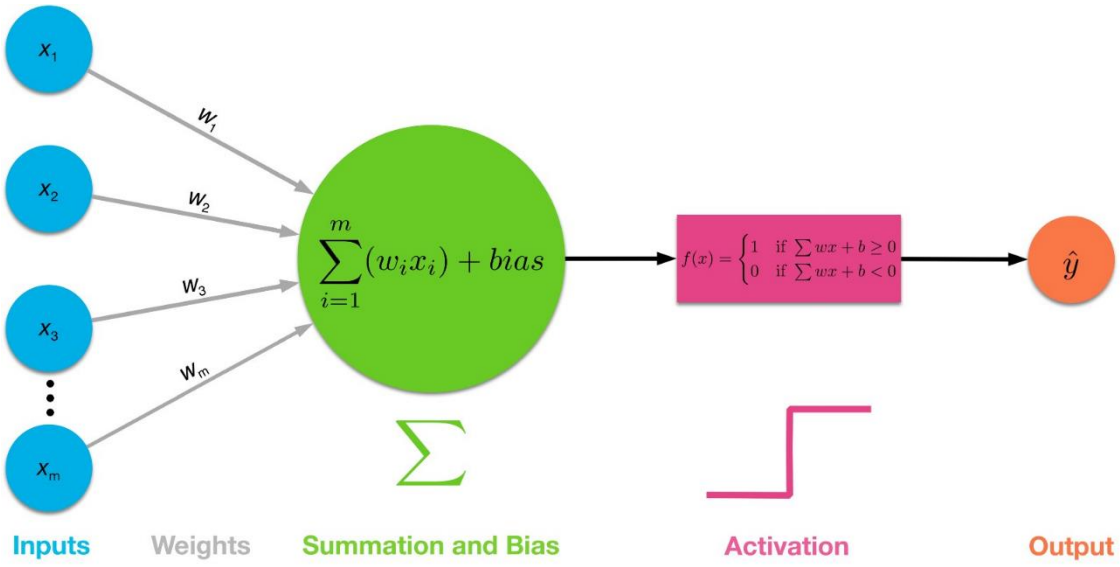

Fig. 3.- Estructura de una red neuronal con una capa de entrada y una de salida.

Al no ser el estudio de la red neuronal la cuestión de este proyecto y haber descrito ya los elementos básicos para poder entender su funcionamiento, concluirá este apartado con el tipo de red que se utilizará en todo el proyecto.

2.1.3.-Tipos de red neuronal y redes del proyecto

Según la forma en que se entrena la red ésta puede tener un aprendizaje supervisado o no supervisado. El entrenamiento supervisado de la red neuronal toma los datos de entrada y de salida, de forma que una vez la red ha sido entrenada y nuevos datos de entrada son introducidos, ésta es capaz de predecir las nuevas salidas de datos. El entrenamiento no supervisado sólo toma los datos de entrada para modelar la estructura o la forma en que se distribuyen, y así aprender más sobre esos datos.

Hay dos tipos de red neuronal en cuanto a entrenamiento supervisado: de clasificación y de regresión. Las redes de clasificación dividen los datos de entrada en diferentes categorías, por ejemplo, dada una población de ensayos de emisiones contaminantes, podría clasificarlos según cumplieran con la normativa y restricciones legales o no. Las redes de regresión predicen un valor para un dato de entrada basándose en la información que tiene.

Las redes utilizadas en este proyecto realizan el entrenamiento supervisado y son de regresión.

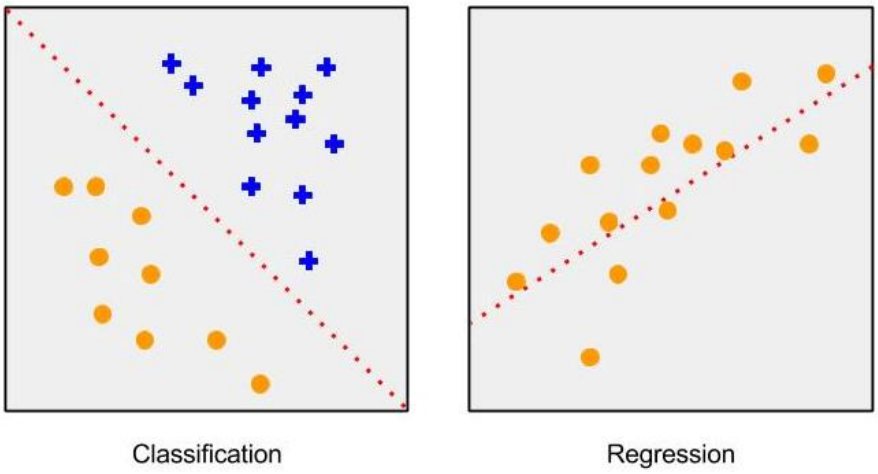

Fig. 4. Tipos de red neuronal con entrenamiento supervisado.

2.2.- Glosario de términos relativos al campo de los motores térmicos

A lo largo del proyecto diversos términos típicos del campo de motores e ingeniería térmica han sido utilizados. Este apartado consiste en un glosario por orden alfabético, donde se definen y explican brevemente esos términos de forma que el lector se familiarice y resulte más sencillo entenderlos en el contexto de este proyecto.

-Comienzo de inyección (SOI – **S**tart **O**f **I**njection): Instante en el que inicia la entrada del combustible en la cámara de combustión.

-Comienzo de energización (SOE – **S**tart **O**f **E**nergizing): Instante en el que se le da la señal al sistema de inyección para que éste comience a inyectar combustible en la cámara de combustión. La diferencia entre el SOE y el SOI es el denominado tiempo de retraso hidráulico del inyector.

-Cierre de válvula de admisión (IVC – **I**ntake **V**alve **C**losing): Instante en el que se cierra la válvula de admisión cortando el paso de aire.

-Fr (dosado de cada pulso, global): El Fr de cada pulso se define como la cantidad de masa inyectada en cada pulso, en relación con el aire total disponible en el cilindro, normalizado por la relación F/A estequiométrica. Sumando la Fr de todos los pulsos, se obtiene el Fr global del motor.

-Fracción másica (de O2, fuel, etc.): Cantidad porcentual que hay de una sustancia o especie en relación al conjunto que hay en el entorno donde esta se encuentra. Por ejemplo, se considera por simplificación en cálculos que en la atmósfera la fracción másica de oxígeno es de 0,23, es decir de todos los gases que forman la atmósfera, el oxígeno supone un 23% del total.

-Pulso de inyección: Cada uno de los eventos de inyección que se realizan en un ciclo. Si un punto de operación tiene inyección piloto, inyección principal y post inyección, en ese caso tendrá tres pulsos.

-Punto muerto inferior (BDC – **B**ottom **D**ead **C**enter): Punto más bajo en el que se encuentra el pistón al final de su carrera descendente dentro del cilindro.

-Punto muerto superior (TDC – **T**op **D**ead **C**enter): Punto más elevado en el que se encuentra el pistón habiendo acabado su carrera ascendente dentro del cilindro.

-Proceso adiabático: Proceso termodinámico en el cual no hay transferencia de calor del sistema hacia cualquier medio o viceversa.

-Retraso de ignición: Cantidad de tiempo que transcurre entre que el combustible diésel se inyecta hasta que sucede la combustión y comienza a inflamarse.

-Sistema de distribución variable: Sistema de distribución que puede variar la regulación de la apertura y el cierre de las válvulas por donde entra el aire (o por donde sale los gases quemados. Con esto se puede aumentar el tiempo en que el cilindro se llena y se vacía, pudiendo optimizarlo para cualquier régimen de giro del motor.

2.3.- Emisiones contaminantes estudiadas en el proyecto

Los mecanismos que controlan las emisiones de hidrocarburos, monóxido de carbono y hollín son menos conocidos que los que controlan a los óxidos de nitrógeno (NO_x). No obstante, todas esas sustancias resultan igual de perjudiciales. Ya existen gran cantidad de estudios y modelos cada vez más fiables para predecir y evitar la contaminación por NO_x. Sin embargo, para estos contaminantes menos conocidos resulta más complicado realizar la misma labor por la naturaleza de las propias emisiones y lo complicado que resulta controlarlas. Por eso es de vital importancia conocerlas y promover el estudio y desarrollo de modelos que ayuden a predecir su formación, para posteriormente evitar y controlar su emisión.

2.3.1.- Partículas en suspensión, hollín (SOOT)

Son partículas de tamaño minúsculo (del orden de unos 25 a 700 nanómetros) compuestas mayoritariamente por carbón, resultantes de una combustión incompleta. Cuanto más pequeñas, más graves pueden ser las consecuencias a su exposición, de modo que diversas leyes regulan el límite de su emisión dependiendo de su tamaño. La principal consecuencia de este tipo de emisiones es un efecto perjudicial en el sistema respiratorio humano: desde irritaciones, a efectos cancerígenos, pues las partículas más pequeñas pueden pasar fácilmente del aire inspirado hasta la sangre.

2.3.2.- Hidrocarburos sin quemar (HC)

Son partículas de elementos procedentes de la combustión de elementos orgánicos cuya aparición se debe a un exceso de combustible en la mezcla a la hora de la combustión. Estos salen directamente a través del tubo de escape a la atmósfera siendo las consecuencias muy variadas y peligrosas. Desde deficiencias respiratorias, neurológicas o metabólicas; hasta alteraciones genéticas, aumento de la mutagénesis y cáncer.

2.3.3.- Monóxido de carbono (CO)

Al igual que el hollín, el monóxido de carbono también es producto de una combustión incompleta. Esta emisión en sí tiene un periodo de vida muy corto y a partir de ella se puede formar dióxido de carbono y ozono. A pesar de ello, contribuye a la destrucción de la capa de ozono, y es peligrosa para los seres vivos, pues se diluye fácilmente en la sangre y reduce la capacidad de transporte de oxígeno en sangre, pudiendo llegar a ser mortal en cantidades excesivas, sobre todo en entornos cerrados.

3.- DESCRIPCIÓN DEL PROYECTO

3.1.- Modelo inicial

En este primer modelo, el proceso para la predicción de emisiones es bastante sencillo. Un fichero de Excel contiene los datos, tanto de entrada como de salida, procedentes de los puntos de operación del motor en condiciones estacionarias. Otro fichero de características similares contiene las condiciones en transitorio. A la hora de introducir los datos en la red se divide la información en tres sets: entrenamiento, test y validación. El primero contiene el 70% de los datos a partir de los cuales la red realizará su entrenamiento. El segundo set (test) contiene 15% de los datos y se utiliza para probar el funcionamiento de la red una vez ha sido entrenada con el primer set de entrenamiento. El tercer set (validación) con los datos restantes, permite comparar durante el entrenamiento si la red diverge mucho de los parámetros iniciales con los que se ha condicionado el entrene para parar el entrenamiento si difieren de forma excesiva. Esto reduce los problemas de overfitting durante el entrenamiento.

Al ejecutar la rutina de MATLAB, se seleccionan los datos de entrada más importantes (que serán comentados en el siguiente apartado) y los de salida (las emisiones contaminantes), y se hace un entrenamiento por separado para cada uno de los contaminantes: HC, CO y SOOT; y así obtener tres redes neuronales independientes.

3.1.1.- Datos introducidos en el modelo

La primera selección de datos de entrada se realizó mediante una selección manual para varias simulaciones del motor, quedando como los parámetros más interesantes los siguientes:

-La presión de inyección medida durante la inyección: es la presión a la que se inyecta el combustible en la cámara de combustión.

-El dosado, o relación que hay entre la masa de combustible y la masa de aire durante el proceso de combustión.

-La velocidad del motor durante el ciclo (revoluciones por minuto).

-La temperatura del agua del motor.

-El retraso de ignición y el final de la combustión.

-Un conjunto de datos relacionados con la fase de cierre de la válvula de entrada (y, por ende, el inicio del ciclo cerrado): La fracción másica de oxígeno, la densidad y la temperatura.

Los datos de salida introducidos fueron la fracción másica de los contaminantes para cada red: YHC, YCO e YSOOT. Los datos se han introducido en la red en forma de logaritmo, pues así los resultados obtenidos son mejores. Cuando se interpreten los resultados, un error de uno sobre cien será equivalente a uno de diez sobre mil. De esta forma, por tanto, cuando se evalúen los errores se hará de forma relativa, no absoluta.

3.1.2.- Resultados del modelo inicial

Los primeros resultados de las redes de HC y CO en este modelo supusieron una mejora con respecto a las correlaciones empíricas. Sin embargo, las predicciones de SOOT sufrían más a causa de una de las características de la red que requería de mejora, la aleatoriedad.

Al ejecutar la rutina que realizaba el entrenamiento de las redes neuronales y daba las predicciones, los casos de HC y CO apenas variaban entre ejecuciones consecutivas. A diferencia de éstas, la red de SOOT daba resultados más dispares, pudiendo incluso predecir algún caso con un error del orden de los millares.

Las siguientes figuras muestran un gráfico donde se relaciona la predicción con los datos reales observados:

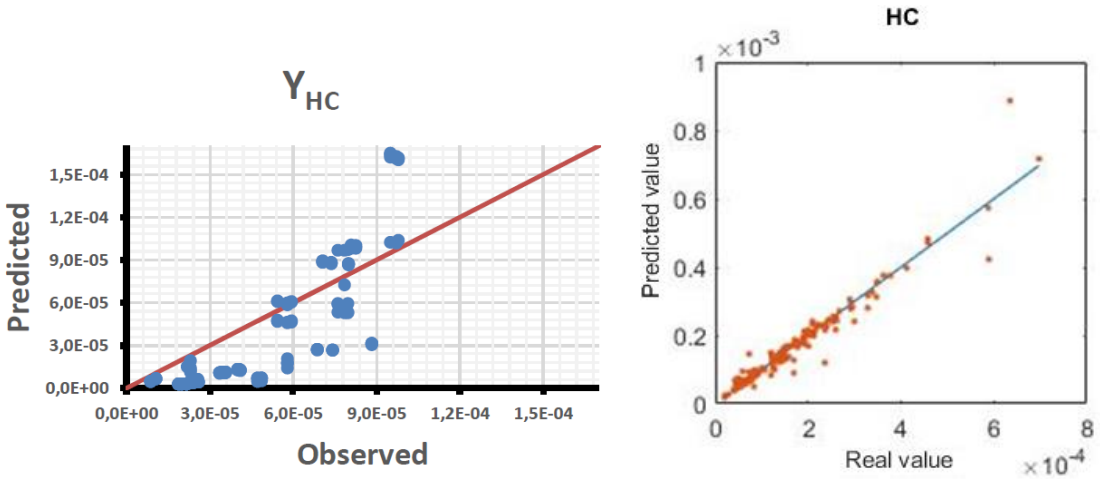

Fig. 5. Comparación de correlaciones empíricas (izquierda) y red (derecha) de HC.

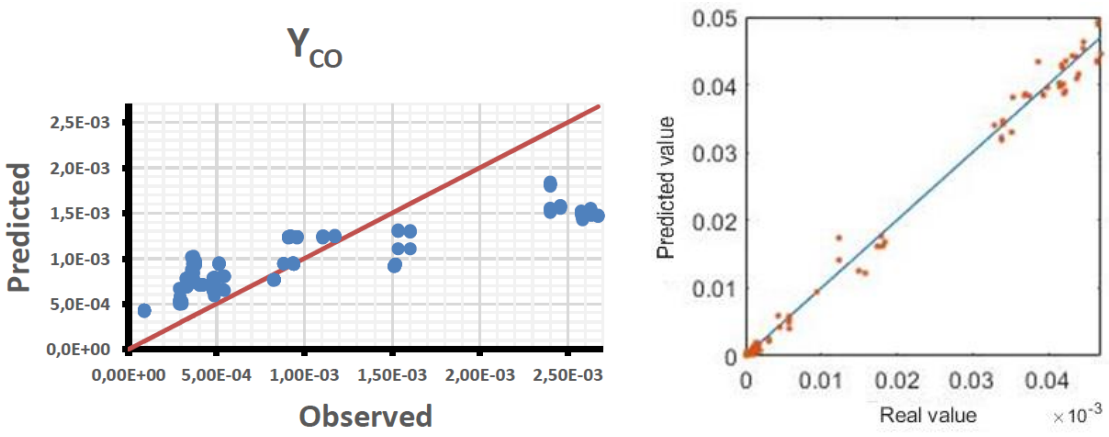

Fig. 6. Comparación de correlaciones empíricas (izquierda) y red (derecha) para el CO.

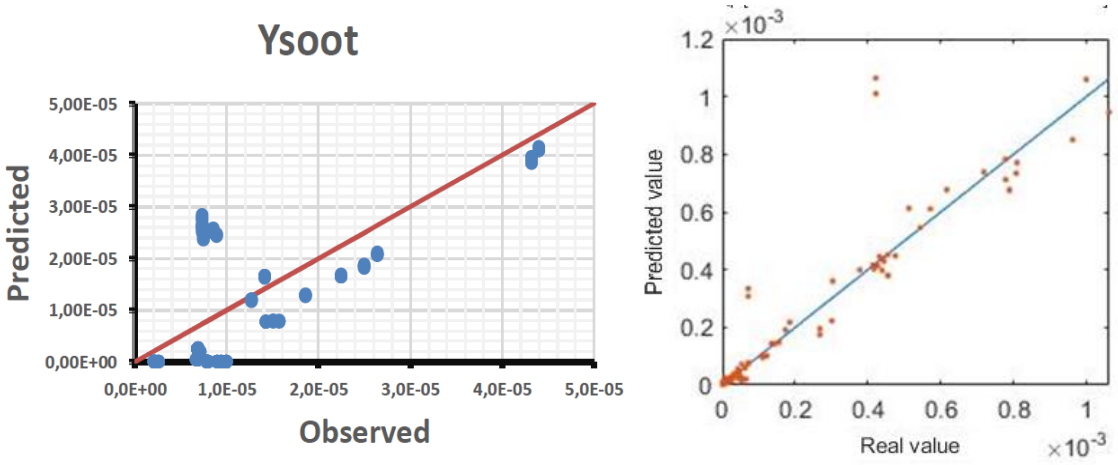

Fig. 7. Comparación de correlaciones empíricas (izquierda) y red (derecha) de SOOT.

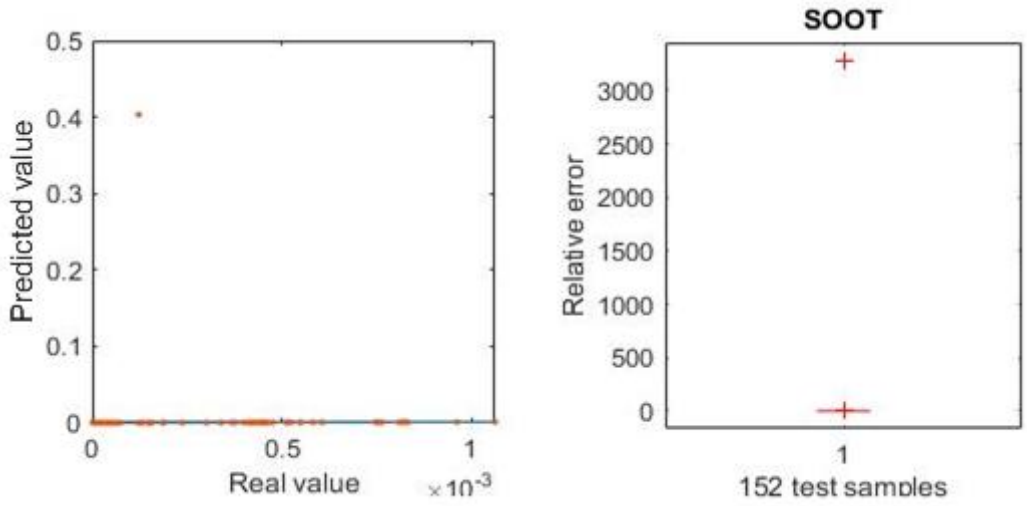

Fig. 8. Problema de predicción en SOOT, dando un error inaceptable.

3.1.3.- Problemas con el modelo

La naturaleza aleatoria del proceso de entrenamiento no fue lo único que ocasionó problemas, surgieron más a la hora de introducir el modelo de predicción en el modelo general del VEMOD.

La aleatoriedad y su fiabilidad:

El principal origen de la aleatoriedad en el entrenamiento de la red viene del conjunto de datos que ésta utiliza para cada ejecución. Los puntos que cogen los grupos de datos de entrenamiento, test y validación, se toman de forma aleatoria, haciendo que con cada entrenamiento los datos de entrada a la red sean distintos, y por consiguiente el entrenamiento realizado sea distinto.

Dado que el proceso que entrena la red y da los resultados se realiza mediante la ejecución manual de una rutina en MATLAB, con cada ejecución el resultado obtenido puede ser más o menos favorable, y eso influye altamente en la fiabilidad. Bien puede obtenerse un resultado adecuado en la primera ejecución que en la quinta o décima, y puede que en una ejecución la red de SOOT dé unos resultados excelentes y la red de CO resultados inaceptables.

Overfitting y underfitting:

Es posible que el entrenamiento de la red pueda llevar a unos resultados sobre ajustados o sin suficiente precisión. Si la red resulta ser muy simple después del entrenamiento no conseguirá buenos resultados pues no ha adquirido el conocimiento suficiente para la predicción, mientras que si está sobre ajustada las predicciones serán 'demasiado precisas' (como se muestra en la figura 9, donde se ve que la función se 'retuerce' para tratar de ajustarse lo mejor posible a los datos experimentales; eso, desgraciadamente, va en detrimento de la capacidad predictiva de la red). El ajuste es tan exacto que al utilizar la red para que realice nuevas predicciones, ésta no es capaz de volver a dar resultados con precisión pues ha memorizado los datos con que se ha modelado.

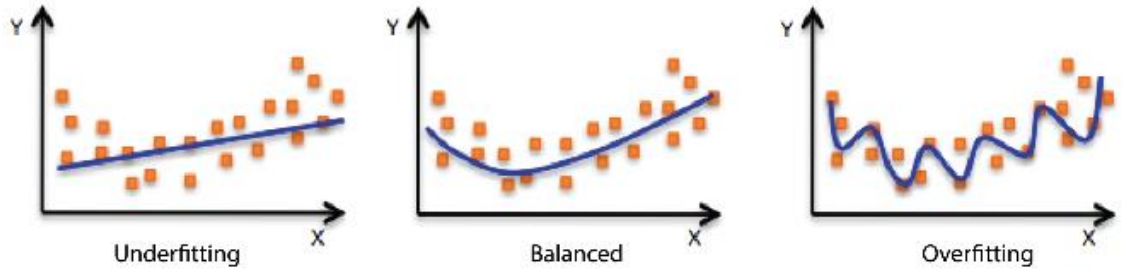

Fig. 9. Comparación del overfitting, underfitting y ajuste correcto.

Emisiones de SOOT en transitorio:

En vista a los resultados obtenidos con la red de SOOT, los datos en transitorio pueden dar algunos problemas como se puede comprobar en la figura 8. A pesar de aportar más precisión puede dar lugar a otros problemas en el modelo general, por lo que convendrá buscar una forma de corregir los datos o tratarlos de manera que pueda evitar esto.

3.2.- Primera fase de mejora

El objetivo de esta primera fase es mejorar la fiabilidad del proceso y solucionar los problemas ocasionados por las emisiones de SOOT, todo ello, gracias a un método con el que se consigue un proceso automático y que no requiere sólo de la intervención y opinión de quien ejecuta la rutina.

3.2.1.- Nuevo método de entrenamiento

Con el nuevo método utilizado para el entrenamiento, la red primero hace un conjunto de entrenamientos previos. Asumiendo una distribución logarítmica-normal para el error de esos primeros casos (que por defecto se ha considerado que con diez casos es suficiente, pero puede ser cualquier número de casos ya que es una de las variables a introducir), se crea una población de errores y a partir de ella se busca un error objetivo. Este error será utilizado como referencia para una segunda fase de entrenamiento. En esta segunda fase la red seguirá entrenando hasta que consiga un error igual o inferior al de referencia, tomando ese caso como la solución. De esta forma se eliminan posibles casos donde el ajuste no haya sido correcto o los resultados sean muy dispares, además de eliminar parte de la subjetividad a la hora de interpretar los resultados.

3.2.2.- Adición de datos de entrada y tratamiento de los datos de SOOT

Algunas observaciones respecto a la selección de datos de entrada se realizaron durante esta fase. En los datos de los que se dispone había casos en los que la estrategia de inyección variaba (en algunos casos había inyección piloto o post inyección). Esto era información que no se le proporcionaba a la red y era útil en la predicción. Para aprovechar esta información se introdujeron los datos del ángulo del cigüeñal donde se produce comienzo de inyección (SOI) y la masa inyectada (Fr) de cada uno de los pulsos, de manera que al sumar los Fr de todos los pulsos se obtendría el Fr global del motor, y quede expresado de forma normalizada.

A la hora de introducir los datos de emisiones de salida de SOOT, un segmento de la rutina de la red se encarga de revisar que no haya datos erróneos o fuera de lugar, ya que los sensores utilizados para medir las emisiones pueden generar algún fallo sistemático debido a las complicadas condiciones en las que tienen que medir. En caso de encontrar un error en algún instante del tiempo, éste sería eliminado, quedando un espacio vacío en el instante de tiempo en que se midió. De manera correspondiente, se creó un set de datos de entrada a parte para cuadrar los datos de entrada con los de salida.

3.2.3.- Resultados con las nuevas mejoras

Con los nuevos cambios la predicción mejoró notablemente. A excepción de algún punto, todos se ajustan casi perfectamente a la recta de regresión. Al graficar los resultados, se añadió a la rutina un segmento que diferenciaría mediante un código de colores los datos de predicción de emisiones en transitorio, estacionario y estacionario con post inyección.

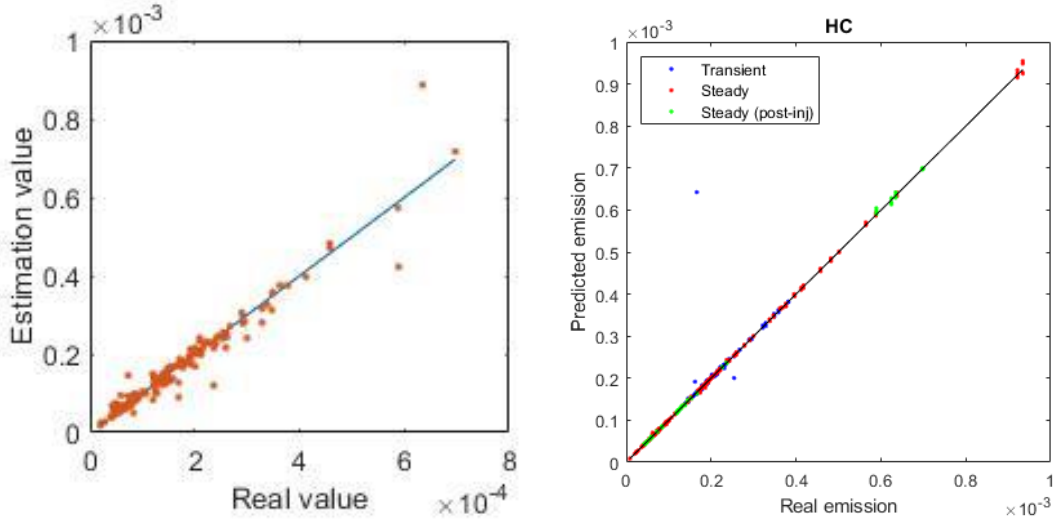

Fig. 10.- Comparación de la red de HC inicial con la primera mejora.

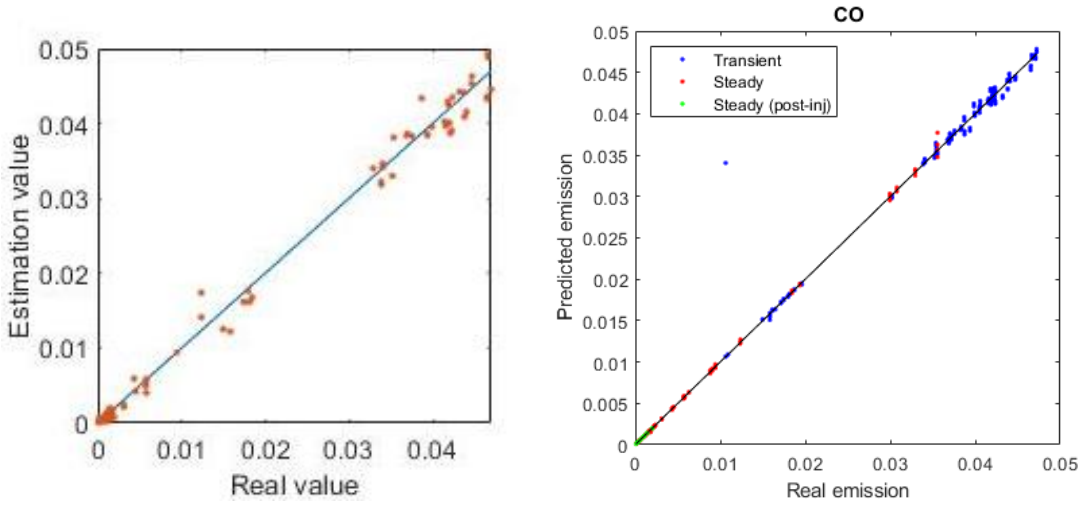

Fig. 11.- Comparación de la red de CO inicial con la primera mejora.

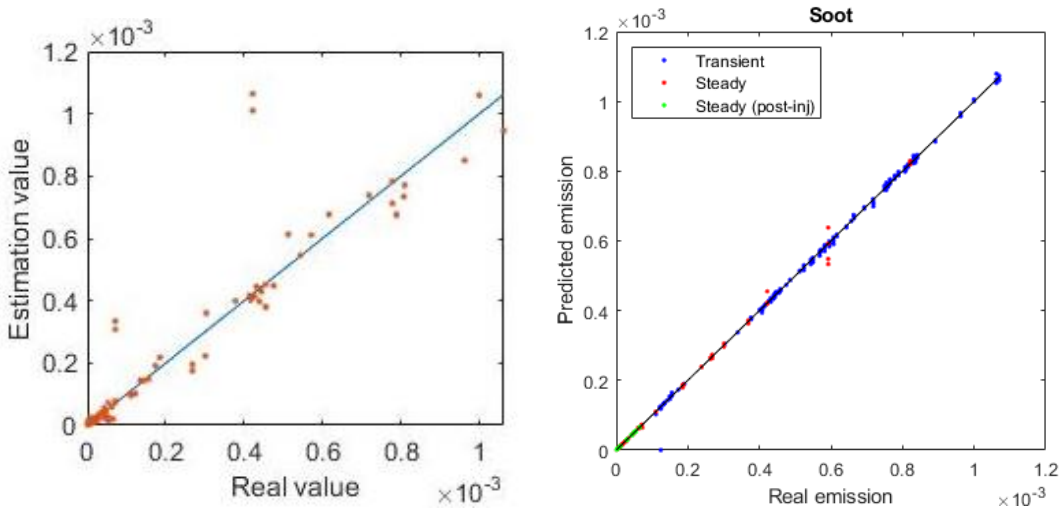

Fig. 12.- Comparación de la red de SOOT inicial con la primera mejora.

3.2.4.- Problemas con la nueva red

Aunque se pensó que todos los problemas relacionados con la fiabilidad y el ajuste de los datos habían sido resueltos, cuando se introdujo de nuevo la red en el modelo y se realizaron pruebas, el modelo comenzó a dar problemas aleatorios por causas desconocidas.

Para comprobar qué sucedía, se inspeccionaron los datos de entrada uno a uno. En algunos de ellos, una pequeña variación (del uno al tres por ciento) en su valor influía drásticamente en los resultados de la predicción. Los resultados variaban mayormente en transitorio.

El problema radica en que la red era 'demasiado compleja' para las predicciones que tenía que realizar. Hasta ahora las redes utilizadas eran de dos capas intermedias y al menos diez neuronas cada una. Cuantas más neuronas y capas forman la red, más parámetros intermedios calcula para dar las predicciones. Si se busca que la red calcule un cierto número de emisiones, y el número de valores intermedios que utiliza para el cálculo doblan o incluso triplican el número parámetros que tiene que calcular, dará muy buenos resultados con esa tanda de datos, pero con otros datos nuevos tendrá problemas (pues habrá overfitting, seguramente). Por tanto, hay que llegar a una solución de compromiso entre simplicidad de la red manteniendo la precisión a la hora de calcular, y robustez, para que pueda trabajar con cualquier grupo de datos que se le presente.

La gran desventaja de que la red sea simple es que los entrenamientos dan en general peores resultados que con una red más compleja. No obstante, puede obtener los mismos o mejores resultados. El factor aleatoriedad tiene más peso en las redes simples.

En la siguiente figura se puede comprobar como ejecutando la misma rutina y realizando el mismo entrenamiento dos veces con una red más simple los resultados varían:

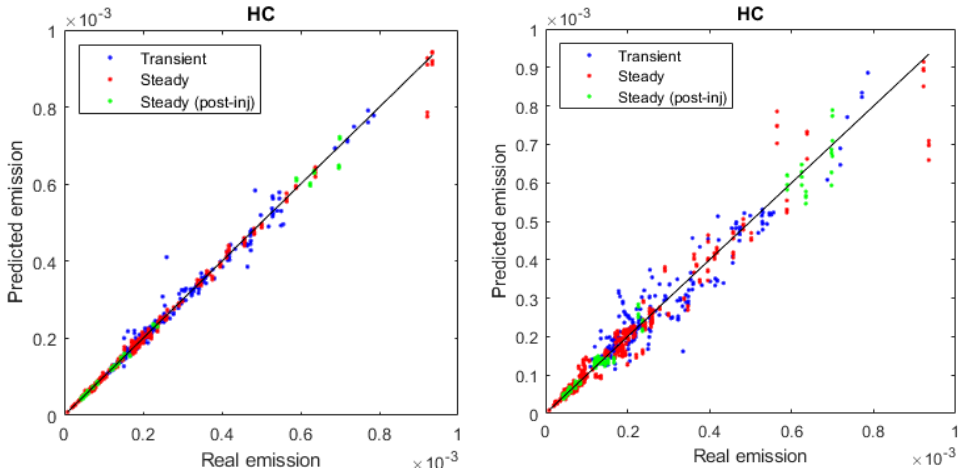

Fig. 13.- Mejores resultados de una red simple en entrenamientos independientes.

3.3.- Fase Final

Esta última fase de mejora resulta ser la más extensa, pues no solo se consiguió solucionar los problemas, sino que se introdujeron diversas mejoras tanto en el código como en el procedimiento seguido por la red, tal y como se comenta en los sucesivos subapartados.

3.3.1.- Creación de una rutina para evaluar la estabilidad de la red

En vista al problema de mejorar la robustez de la red se decidió crear una rutina para poder comprobar el comportamiento del mejor de los casos durante el entrenamiento ante un cambio en sus datos de entrada.

El funcionamiento de esta rutina se basa en observar lo que predice una red entrenada con un set de datos de entrada estándar, y compararlo con lo que prediga con un segundo conjunto de datos perturbados. Lo que difiere ese grupo de datos del primero es que se ha introducido una pequeña variación en una de las entradas, simulando una posible perturbación a la hora de la medición o de la toma aleatoria de datos por parte de la red. Por defecto la variación es de un tres por ciento y se introduce en la entrada de la fracción másica de oxígeno ya que de todas las entradas (temperatura, densidad, comienzo de inyección, etc.), se comprobó que ésta era la más sensible en cuanto a la predicción.

Una vez se han obtenido resultados con ambos sets de datos se procede a realizar una comparación entre ellos. Tomando como referencia los datos originales, se calcula el error relativo de los datos perturbados para cuantificar en tanto por cien la diferencia. También se calcula el coeficiente de regresión entre ambos resultados y se representan gráficamente para poder comparar los resultados con mayor facilidad.

En la siguiente figura (Fig. 15) se demuestra lo severo que puede ser el efecto de esa pequeña variación en unas gráficas donde se comparan para cada contaminante los resultados obtenidos con los datos normales y con los datos perturbados. Para una buena red con baja sensibilidad ante cabios en los datos, los puntos se ajustarían correctamente a la recta de regresión. En el caso de las emisiones de SOOT la diferencia tan grande explicaría el porqué de los errores inesperados en la red.

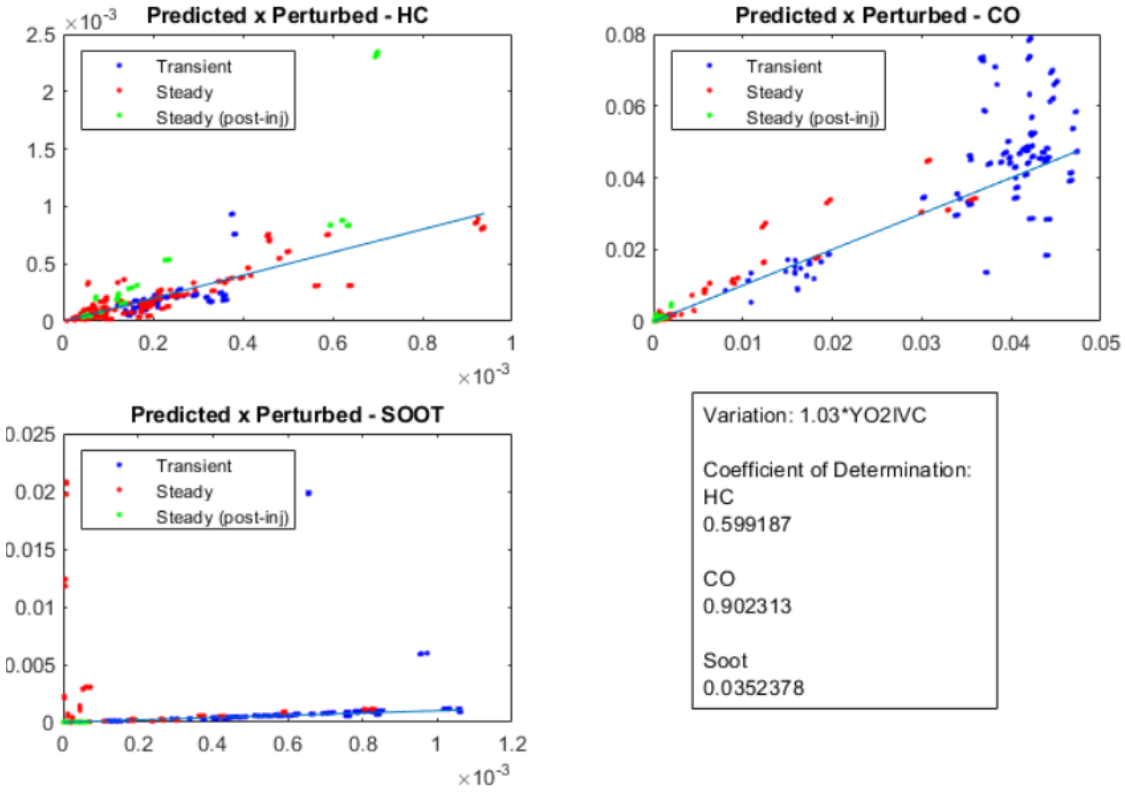

Fig. 15. Resultados de la red de cada contaminante introduciendo los datos con perturbación vs introduciendo los datos sin perturbación.

3.3.2.- Cambio en los datos introducidos en la red

Nuevas discusiones sobre la sensibilidad de los datos de entrada en la red llevaron a cuestionar el hecho de que algunos datos se den en condiciones de cierre de válvula (IVC). Al tratarse de un motor equipado con un sistema de distribución variable, que modifica la apertura y el cierre de las válvulas para aumentar o disminuir el tiempo de llenado y vaciado del cilindro según la velocidad de giro del motor, ello influiría en la referencia para la predicción de la red. Para evitar esa variabilidad, se pensó en tomar esos datos respecto a un punto que resultara más invariante. Las dos opciones que se disponían era tomarlos respecto al punto muerto inferior (BDC) o el punto muerto superior del cilindro (TDC).

En la figura 16 se pretende ilustrar, de manera esquemática, la relevancia de tomar esta referencia fija. En ella se muestran dos curvas de presión en cilindro para dos casos en los que se ha variado el IVC. En ambos casos, no obstante, se tiene una misma evolución de la presión durante el ciclo cerrado. Dado que las condiciones en el ciclo cerrado son las mismas, la red debería predecir los mismos contaminantes. Sin embargo, como las condiciones al cierre (IVC e IVC') son diferentes, eso no será así. Si las condiciones que se dan como entrada a la red son las que tienen lugar en el TDC, entonces sí que tendríamos las mismas entradas a la red en ambos casos, y por tanto las predicciones serían las mismas.

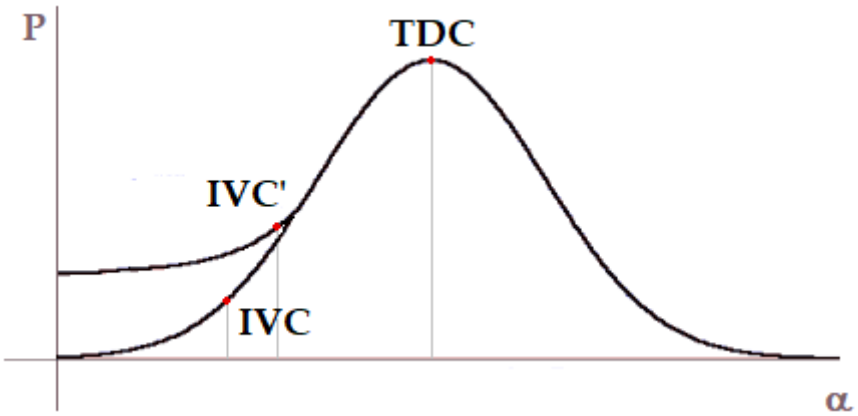

Fig. 16. Variabilidad del cierre de válvula.

Conociendo las características de motor con una simple conversión mediante la ecuación de los procesos adiabáticos (considerando como coeficiente adiabático γ=1.4) y una relación de volúmenes, se puede cambiar la temperatura, densidad y presión de IVC a TDC o BDC. Se decidió convertir los datos respecto al punto muerto superior para de esa manera darle a la red la posibilidad de tener sensibilidad a eventuales cambios en la relación de compresión del motor:

$$\rho_{TDC} = \rho_{IVC} * \frac{V_{IVC}}{V_{TDC}}\ ;\quad T_{TDC} = T_{IVC} * (\frac{V_{IVC}}{V_{TDC}})^{\gamma-1}\ ;\quad P_{TDC} = P_{IVC} * (\frac{V_{IVC}}{V_{TDC}})^{\gamma}\ ;$$

Donde ρ_{TDC} y ρ_{IVC} ; T_{TDC} y T_{IVC}; y P_{TDC} y P_{IVC}, son las densidades, temperaturas, y presiones en el punto muerto superior y en el IVC, respectivamente, y γ el coeficiente adiabático.

También se cambiaron los datos del comienzo de inyección (SOI) por los mismos motivos, utilizando el instante en que se da la señal al inyector para que inyecte el combustible (SOE) en vez del instante en que inicia realmente el proceso de inyección (SOI), puesto que el SOE es el valor que realmente se conoce, al ser el parámetro que controla la unidad de control electrónico del motor.

3.3.3.- Nuevo método de entrenamiento más adecuado

Tras observar que el mejor de los casos podía llevar a resultados poco fiables debido a la sensibilidad de la red, era necesario crear modificaciones en la rutina para no guardara sólo la información del 'mejor' de los entrenamientos.

Se decidió generar una función cada vez que la red terminara un entrenamiento, de forma que cuando se llegara al caso donde el error era inferior al de los diez entrenamientos previos, toda la información de los entrenamientos quedaría almacenada y organizada para posteriormente poder abrirla y evaluar su sensibilidad. Se ejecutó la nueva rutina y se estudiaron los resultados obtenidos hasta llegar a la conclusión de que no se conseguía una solución comprometida entre precisión y sensibilidad. Era difícil encontrar un caso que se adaptara a las condiciones que se estaban buscando.

El problema era una cuestión de probabilidad. Cuantos más casos se dispusieran, más probabilidad de que uno de ellos fuera preciso y suficientemente robusto. Aumentando el número de entrenamientos previos a realizar se obtendría un error de referencia más pequeño, ya que con una población más grande eso es más probable. De esta forma, cuando la red entrena para obtener la predicción, el número de casos que se obtienen hasta que ésta alcanza un error inferior al error previo establecido es mucho mayor. Sin embargo, aun siendo el método más adecuado en la teoría, el problema surgió con la primera ejecución de esta nueva rutina. El error obtenido con los casos previos puede ser tan pequeño que cuando la red entrene no pueda alcanzarlo, o pasen horas hasta que se llegue a un error inferior.

3.3.4.- Rutina final y resultados

La rutina final se basa en el método del error pequeño explicado en el apartado anterior combinado con la rutina de evaluación de sensibilidad, pero limitando el número de entrenamientos que realiza la red. Se ha introducido una variable para que el usuario decida el número límite de entrenamientos a realizar por la red antes de ejecutar la rutina. Por defecto se ha considerado que con cien casos es suficiente para que al menos uno de ellos sea suficientemente preciso y sensible a pequeñas perturbaciones en los datos. Además, para no desaprovechar nada de información durante el proceso, todos los entrenamientos previos para caracterizar el error son guardados también en forma de función por si uno de ellos resulta ser mejor.

Para que la decisión del mejor caso sea más sencilla y visual, la rutina genera un fichero de Excel donde se grafica la precisión de la red y la sensibilidad tanto de los entrenamientos previos como de los entrenamientos definitivos. El eje de las ordenadas es el coeficiente de regresión y el de abscisas el porcentaje medio de diferencia entre la predicción utilizando los datos con perturbación y sin perturbación.

El punto con mayor coeficiente de regresión (R^2) y menor diferencia media entre resultados (Error medio de estabilidad), corresponderá con el mejor entrenamiento.

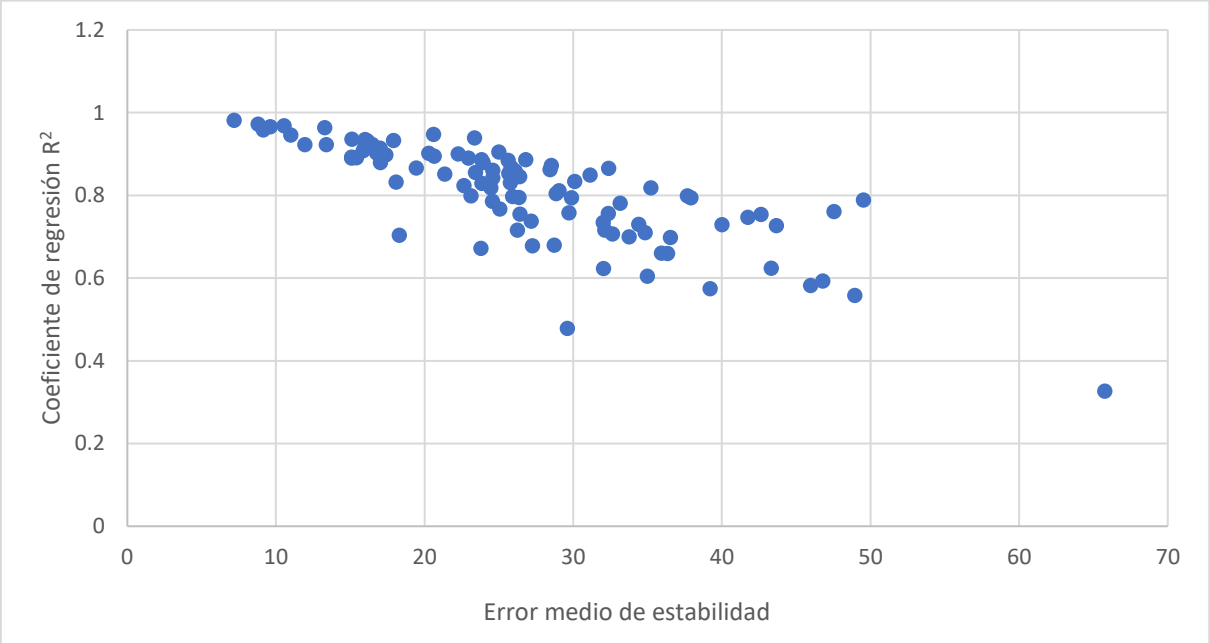

Fig. 17. Grafica de las características clave de cada entrenamiento de la red de HC.

Para observar los resultados de la rutina final, ésta se ha ejecutado y se ha decidido manualmente, con las gráficas de Excel para visualizar el mejor caso para cada emisión, cuál de los entrenamientos ha sido el más preciso y a la vez el más estable en caso de una perturbación o variación en sus datos.

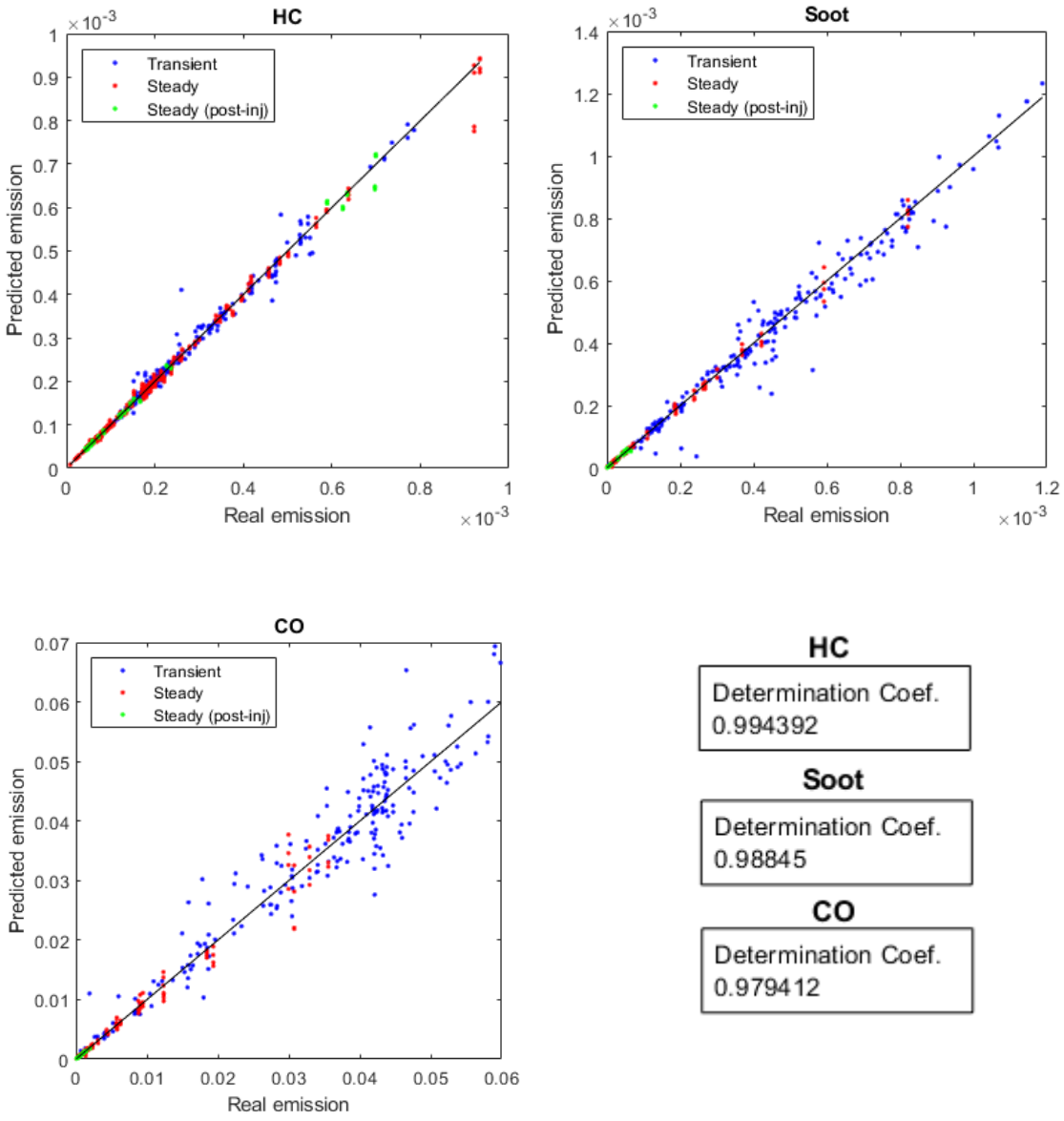

Fig. 18. Resultados de las predicciones de la red final.

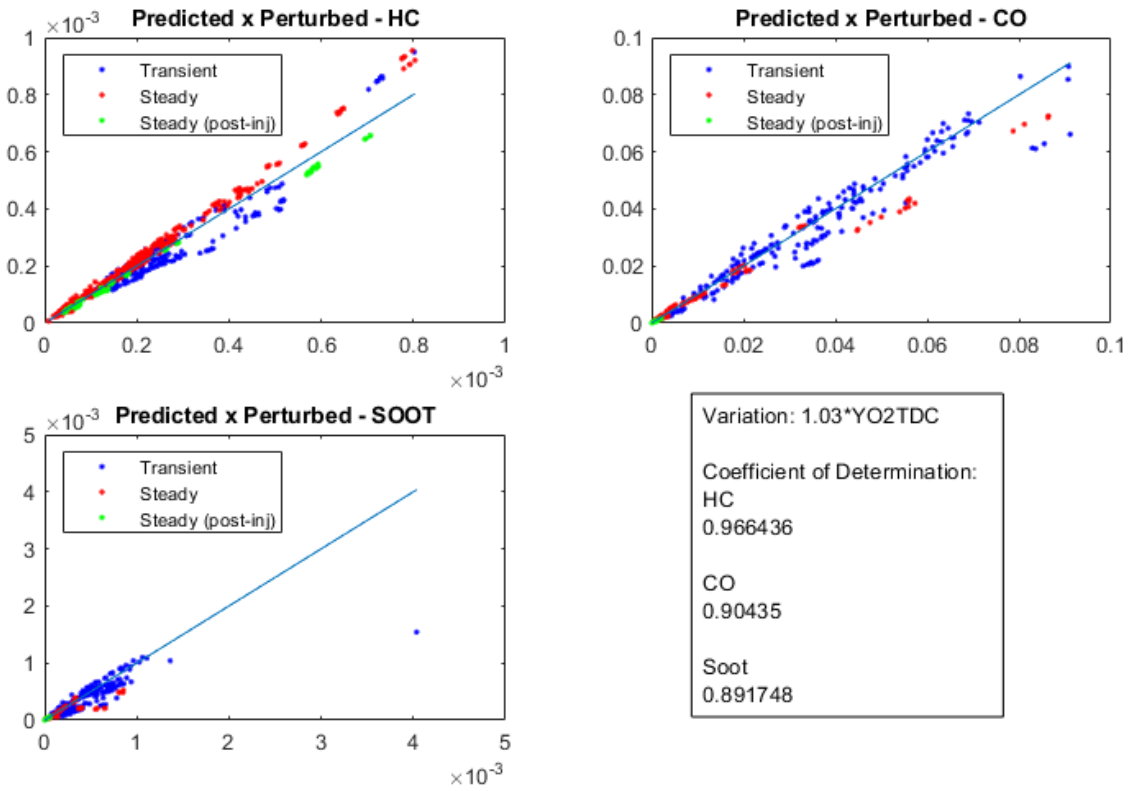

Fig. 19. Comparación de resultados con perturbación y sin perturbación en las redes finales.

3.4.- Evolución y comparación de las fases de la red

Para observar la evolución y la mejora, en este apartado se comparan los resultados obtenidos por cada una de las fases que la red de HC ha conseguido a lo largo del proyecto (Fig.20).

Respecto a la primera red neuronal que se creó para HC, no sólo destaca la mejora en precisión, sino también la mejora de la calidad y la diferenciación de las condiciones de transitorio y estrategias de inyección a la hora de graficar los puntos para observar qué condiciones han sido más difíciles de predecir para la red.

Teniendo en cuenta que en las dos primeras fases la red neuronal era de dos capas ocultas con al menos diez neuronas en cada una, la red final de una sola capa oculta ha conseguido muy buenos resultados. A pesar de haber cedido parte de la precisión a la hora de obtener las predicciones en el desarrollo entre la primera fase de mejora y la fase final, la estabilidad que se ha conseguido hace insignificante esta pérdida en comparación.

El resultado final es una red altamente precisa y estable, es decir, robusta ante pequeñas perturbaciones en sus datos.

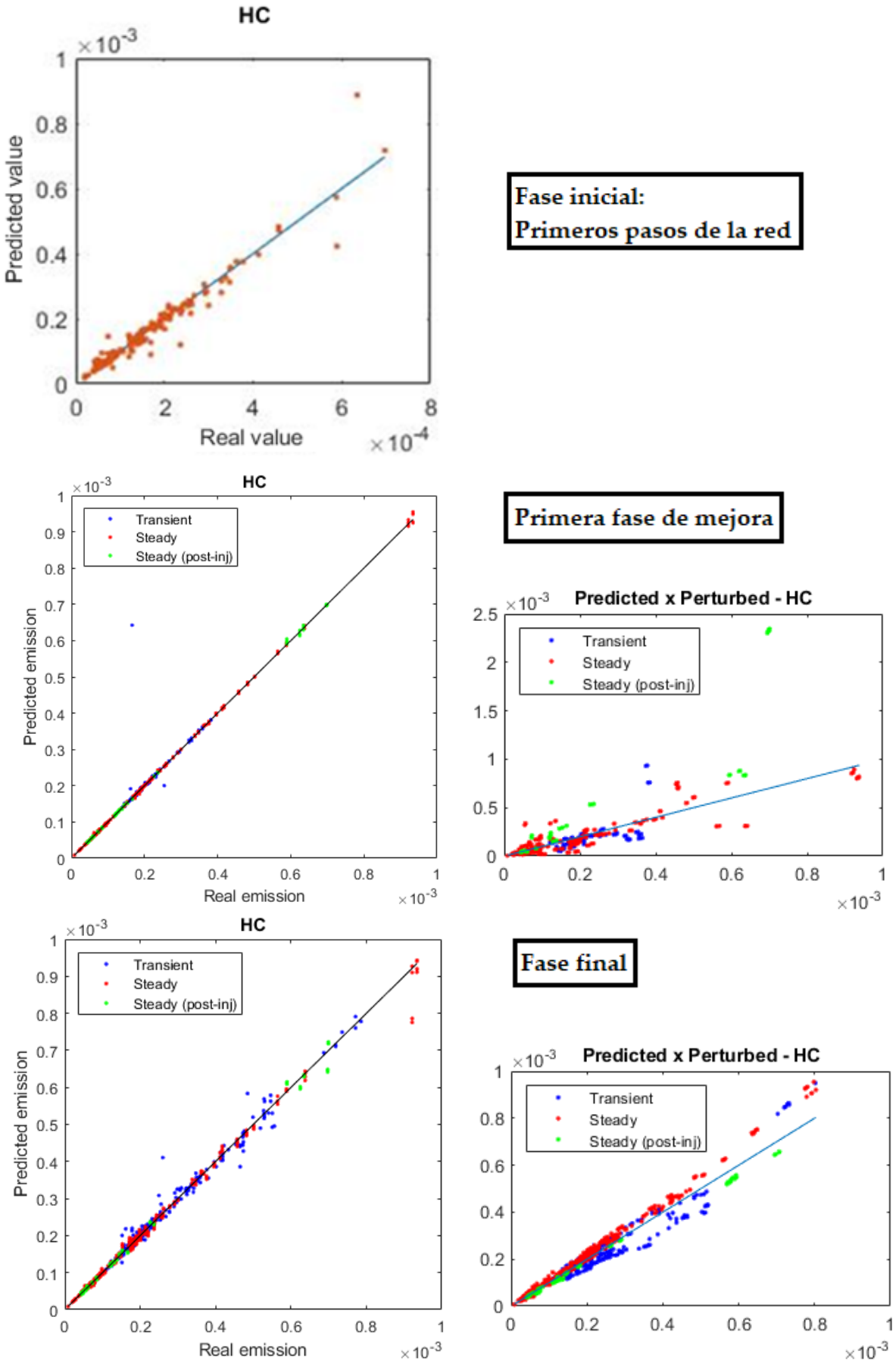

Fig. 20. Evolución fase a fase de la red de HC

4.- Conclusión

Al comienzo del proyecto fue necesario familiarizarse con el lenguaje de programación MATLAB, y sobre todo con el campo teórico y de investigación sobre redes neuronales artificiales para entender su funcionamiento, su situación actual de desarrollo, y su uso en el contexto del sector de motores y la predicción de emisiones. Con todos esos conocimientos nuevamente adquiridos y el bagaje de ingeniería térmica y estadística, adquirido durante el grado, se consiguió desarrollar una nueva fase de la red neuronal a donde la precisión mejoró notablemente, y se solucionaron los problemas que presentaba el modelo predecesor en cuanto a las emisiones de SOOT.

El verdadero desafío surgió cuando en esa fase que se creía perfeccionada comenzaron a surgir errores en el submodelo de emisiones del VEMOD. A pesar de haber ingeniado cambios en los datos de entrada de la red para que pudiera predecir respecto a unas referencias más adecuadas con objeto de solucionarlo, seguían apareciendo algunos errores. En ese momento se descubrió lo importante que era la robustez de la red, es decir, el evitar su excesiva sensibilidad para que no haya una gran variación en sus predicciones cuando se encuentre con una pequeña variación en los datos de entrada.

El modelo de la red final consiguió lidiar con estos problemas mediante un nuevo método para el entrenamiento, y la introducción de una rutina crucial para la evaluación de la sensibilidad. Con este método, se aprovecha y muestra toda la información generada durante el proceso de cada entrenamiento. Posteriormente se escoge manualmente aquél que ha conseguido la mejor solución de compromiso entre precisión a la hora de predecir, y sensibilidad y capacidad de respuesta.

Para futuras investigaciones, un aspecto a mejorar del modelo final sería la automatización del proceso de selección. Conociendo ahora el potencial de las redes neuronales, se podría incluso desarrollar una pequeña red secundaria para que interprete los casos y seleccione por sí misma el más conveniente.

El periodo de tiempo en que se ha trabajado para el desarrollo de estas redes neuronales ha culminado en un exitoso resultado final, y en un gran aprendizaje en el campo de los motores de combustión interna alternativos, programación, y machine learning para ANN. Se puede decir con seguridad que el modelo de predicción generado por este proyecto tendrá un impacto positivo no sólo en el submodelo de combustión y emisiones, sino en el modelo general del VEMOD.

5.- Presupuesto

Este apartado consta de una descripción detallada del coste que ha supuesto la investigación y el desarrollo del proyecto "Desarrollo de un modelo basado en redes neuronales artificiales para la predicción de emisiones contaminantes en un motor diésel DI". El tiempo total invertido por el ingeniero para la realización de este trabajo es de 480 horas

Los costes del proyecto se pueden desglosar en costes materiales, costes de software, y costes de mano de obra, según su naturaleza.

5.1.- Costes materiales

En este apartado quedan detallados los costes de las herramientas y productos materiales utilizados durante el proyecto.

Para el cálculo de estos costes de utilizará la fórmula financiera de la amortización:

$$Valor_{amortización} = \frac{V_{ad} - V_r}{T_a}$$

Donde V_{ad} es el valor de adquisición del producto en euros, V_r el valor residual que tendrá el producto al acabar su vida útil, también en euros, y T_a el periodo o tiempo en que se desea amortizar el producto en horas.

El material utilizado por el ingeniero ha sido un ordenador portátil Lenovo IdeaPad 320 con un valor de adquisición de 750 euros, y sin valor residual; y un conjunto de torre computadora, monitor, teclado y ratón MSI con un valor de adquisición de 650 euros, y un valor residual de 150 euros. Se considera el tiempo en que se desea amortizar el material de dos años, y un mes de trabajo de 120 horas.

La amortización material será entonces:

$$Amortización_{portátil} = \frac{750-0}{24\ x\ 120} = 0{,}26\ €/h$$

$$Amortización_{equipo} = \frac{650-150}{24\ x\ 120} = 0{,}17\ €/h$$

Quedan resumidos los gastos asociados a material en la siguiente tabla:

Concepto	Precio de amortización	Tiempo de uso	Precio
Ordenador portátil	0,26 €/h	480 h	124,00 €
Equipo fijo	0,17 €/h	480 h	81,60 €
		Precio total	205,60 €

5.2.- Costes de software

Para la realización de este proyecto es imprescindible el uso de la suite de Office y el programa Matlab. Todo lo relacionado con el desarrollo, desde los ajustes en los datos de entrada hasta la creación de la propia red neuronal se ha programado desde Matlab. El programa de hojas de cálculo Excel se requiere para grabar y clasificar los datos y resultados.

El precio de la licencia anual de Matlab es de 800 €, y el de la suite de office de 69 €, ambos sin valor residual. Utilizando la m isma fórmula para el cálculo de la amortización del apartado anterior y considerando el tiempo en que se desea amortizar la licencia de Office el periodo en que el ingeniero ha trabajado en el proyecto (4 meses), el coste de amortización de software será:

$$Amortización_{Office} = \frac{69}{4\ x\ 120} = 0{,}14\ €/h$$

En la siguiente table se resume los costes de software:

Concepto	Precio de amortización	Tiempo de uso	Precio
Licencia anual Matlab	-	-	800,00 €
Licencia Office	0,14 €/h	480 h	69,00 €
		Precio total	869.00 €

5.3.- Costes de mano de obra

Los costes que han supuesto el trabajo realizado por las personas involucradas en el proyecto se contemplan en este apartado.

-Ingeniero junior: El ingeniero junior se ha encargado, además del desarrollo de la red neuronal, del tratamiento y procesado de los datos de entrada para la red neuronal y de la redacción de la memoria del proyecto. El precio por hora del trabajo del ingeniero junios se considera de 15€/h.

-Director y tutor de proyecto: Ha ayudado al ingeniero a lo largo del proyecto mediante tutorías y ha supervisado que el trabajo realizado fuera correcto. Se ha considerado el precio por hora del director y tutor de proyecto de 35€/h.

Los gastos de mano de obra quedan resumidos en la siguiente tabla:

Concepto	**Precio de mano de obra por hora**	**Tiempo de uso**	**Precio**
Ingeniero			
Procesado y tratamiento de datos	15 €/h	100 h	1.500 €
Desarrollo de la red neuronal	15 €/h	300 h	4.500 €
Redacción de la memoria	15 €/h	80 h	1.200 €
Director y tutor de proyecto			
Supervisión de tareas y tutelaje	35 €/h	40 h	1.400 €
		Precio Total	8.600 €

5.3.-Presupuesto final

Finalmente, se procede a agrupar todos los gastos para obtener el presupuesto total del proyecto.

Concepto	Precio de amortización / Mano de obra	Tiempo de uso	Precio
Material			
Ordenador portátil	0,26 €/h	480 h	124,00 €
Equipo fijo	0.17 €/h	480 h	81,60 €
Software			
Licencia anual Matlab	-	-	800,00 €
Licencia Office	0,14 €/h	480 h	69,00 €
Ingeniero			
Procesado y tratamiento de datos	15,00 €/h	100 h	1.500,00 €
Desarrollo de la red neuronal	15,00 €/h	300 h	4.500,00 €
Redacción de la memoria	15,00 €/h	80 h	1.200,00 €
Director y tutor de proyecto			
Supervisión de tareas y tutelaje	35,00 €/h	40 h	1.400,00 €
		Subtotal	9674,60 €
Costes indirectos (2%)			193,49 €
IVA (21%)			2.031,66 €
		PRECIO TOTAL	11.899,76 €

El presupuesto final del proyecto asciende a un total de once mil ochocientos noventa y nueve euros con setenta y seis céntimos.

6.-Referencias.

Martin, J., Arnau, F., Piqueras, P., and Auñon, A., "Development of an Integrated Virtual Engine Model to Simulate New Standard Testing Cycles," SAE Technical Paper 2018-01-1413, 2018, doi:10.4271/2018-01-1413.

Apuntes de Combustión de tercer curso de grado en ingeniería mecánica.

Alonso, J.M., Alvarruiz, F., Desantes, J.M., Hernandez, L., Hernandez, V., Molto, G. (2007). "Combining Neural Networks and Genetic Algorithms to Predict and Reduce Diesel Engine Emissions", IEEE Transactions on Evolutionary Computation.

B. Krose, P. van der Smagt, "An introduction to Neural Networks", The University of Amsterdam, eight edition, 1996.

Guardiola, C., Ruiz, S., Pla, B., Blanco-Rodriguez, D., "Motores térmicos para ingeniería mecánica".

Apuntes de Motores Térmicos de tercer curso de grado en ingeniería mecánica.

Hector G. Amino, "Development of an empirical pollutant emission predictive model". Master Thesis, 2018.

Anexo: Código en Matlab de la red final.

```
%% Data lecture
    % Steady and transient data lecture. Be sure that the xls files are in the current used
    % folder.

clear; clc; close all;
[valSteady, texte1, tabentier1]=xlsread('Steady_final_NEW.xlsx','OutputANN5');

[valtr, txt]=xlsread('transient_imputs.xlsx','Ambient');
[valtrC, txtC]=xlsread('transient_imputs.xlsx','Cold');
[valtrH, txtH]=xlsread('transient_imputs.xlsx','Hot');

Heading={'Pinj' 'YO2_IVC' 'Rho_TDC' 'N' 'T_TDC' 'T_w' 'SOE1' 'FrPulse1' 'SOE2' 'FrPulse2' 'SOE3' 'FrPulse3' 'SOE4' 'FrPulse4' 'Exp' 'Pred' 'rel_error (%)' 'mean_rel_error (%)' 'Pred_var' 'rel_error_var (%)' 'mean_rel_error_var (%)' };
% When writing the output file


Version='2.0';

%% Setting some parameters

training_iterations = 100; %number of iterations to stablish the error limit (MSE of validation process)
Z_target = -1.64; %  5% %take values whithin 5% or less of normal distribution
%Z_target = -1.40; %  8%
%Z_target = -1.28; % 10%
%Z_target = -1.17; % 12%
%Z_target = -0.2; % rapid


%% Reading Steady data

%# Outputs to ANN

YHC=valSteady(:,9);
YCO=valSteady(:,10);
YSOOT=valSteady(:,19);

%# Finding and removing any erroneous data (i.e. zero values)

posYSOOT01 = find(YSOOT==0); %returns the position of zero values
posYSOOT02 = find(YSOOT~=0); %returns the position of non-zero values
YSOOT(posYSOOT01)=[];        %exclude the zero values from YSOOT

%# Inputs to ANN

ID=valSteady(:,2);
TTDC=valSteady(:,4);
N=valSteady(:,5);
Tw=valSteady(:,20);
Pinj=valSteady(:,11);
Fr=valSteady(:,14);
YO2IVC=(valSteady(:,15));
PTDC=valSteady(:,29);
EOC=valSteady(:,17);
rhoTDC=valSteady(:,18);
SOE1=valSteady(:,21);
FrPulse1=valSteady(:,22);
SOE2=valSteady(:,23);
```

```
FrPulse2=valSteady(:,24);
SOE3=valSteady(:,25);
FrPulse3=valSteady(:,26);
SOE4=valSteady(:,27);
FrPulse4=valSteady(:,28);


 %% Reading Transient data

%# Outputs to ANN

YHCt=valtr(:,22);
YCOt=valtr(:,21);
YSOOTt=valtr(:,23);

%Data from the new transients
%cold
YHCtC=valtrC(:,22);
YCOtC=valtrC(:,21);
YSOOTtC=valtrC(:,23);
%hot
YHCtH=valtrH(:,22);
YCOtH=valtrH(:,21);
YSOOTtH=valtrH(:,23);


%# Finding and removing any erroneous data (i.e. zero values) (ambient data)
posYSOOTt01 = find(YSOOTt==0);
posYSOOTt02 = find(YSOOTt~=0);
if posYSOOTt01 ~= 0
YSOOTt(posYSOOT01)=[];
end

%# Cold data
posYSOOTtC01 = find(YSOOTtC==0);
posYSOOTtC02 = find(YSOOTtC~=0);
if posYSOOTtC01 ~= 0    % If there are no zeros, continue
YSOOTtC(posYSOOTtC01)=[];
end

%# Hot data
posYSOOTtH01 = find(YSOOTtH==0);
posYSOOTtH02 = find(YSOOTtH~=0);
if posYSOOTtH01 ~= 0    % If there are no zeros, continue
YSOOTtH(posYSOOTtH01)=[];
end


  %# Inputs to ANN

%IDt=valtr(:,2); % not necessary

TTDCt=valtr(:,18);
Nt=valtr(:,2);
Twt=valtr(:,3);
Pinjt=valtr(:,16);
Frt=valtr(:,15);
YO2IVCt=valtr(:,17);
PTDCt=valtr(:,20);
rhoTDCt=valtr(:,19);
```

```
SOE1t=valtr(:,4);
FrPulse1t=valtr(:,5);
SOE2t=valtr(:,6);
FrPulse2t=valtr(:,7);
SOE3t=valtr(:,8);
FrPulse3t=valtr(:,9);
SOE4t=valtr(:,10);
FrPulse4t=valtr(:,11);

 %# imputs for the other two transients.

%Cold
TTDCtC=valtrC(:,18);
NtC=valtrC(:,2);
TwtC=valtrC(:,3);
PinjtC=valtrC(:,16);
FrtC=valtrC(:,15);
YO2IVCtC=valtrC(:,17);
PTDCtC=valtrC(:,20);
rhoTDCtC=valtrC(:,19);
SOE1tC=valtrC(:,4);
FrPulse1tC=valtrC(:,5);
SOE2tC=valtrC(:,6);
FrPulse2tC=valtrC(:,7);
SOE3tC=valtrC(:,8);
FrPulse3tC=valtrC(:,9);
SOE4tC=valtrC(:,10);
FrPulse4tC=valtrC(:,11);

%Hot
TTDCtH=valtrH(:,18);
NtH=valtrH(:,2);
TwtH=valtrH(:,3);
PinjtH=valtrH(:,16);
FrtH=valtrH(:,15);
YO2IVCtH=valtrH(:,17);
PTDCtH=valtrH(:,20);
rhoTDCtH=valtrH(:,19);
SOE1tH=valtrH(:,4);
FrPulse1tH=valtrH(:,5);
SOE2tH=valtrH(:,6);
FrPulse2tH=valtrH(:,7);
SOE3tH=valtrH(:,8);
FrPulse3tH=valtrH(:,9);
SOE4tH=valtrH(:,10);
FrPulse4tH=valtrH(:,11);


%% SET ALLDATA
    % Array used for data division.

    % .. Steady .. %
ALLDATA(1,:)=(Pinj);
ALLDATA(2,:)=(YO2IVC);
ALLDATA(3,:)=(rhoTDC);
ALLDATA(4,:)=N;
ALLDATA(5,:)=(TTDC);
ALLDATA(6,:)=(Tw);
ALLDATA(7,:)=(SOE1);
ALLDATA(8,:)=(FrPulse1);
```

```
ALLDATA(9,:)=(SOE2);
ALLDATA(10,:)=(FrPulse2);
ALLDATA(11,:)=(SOE3);
ALLDATA(12,:)=(FrPulse3);
ALLDATA(13,:)=(SOE4);
ALLDATA(14,:)=(FrPulse4);
%ALLDATA(15,:)=(PTDC*10^5); % Added TDC Pressure in steady case (not used)
%ALLDATA(16,:)=(Fr);
%ALLDATA(17,:)=(ID);
%ALLDATA(18,:)=EOC;


    % .. Transient .. %
ALLDATAt(1,:)=(Pinjt);
ALLDATAt(2,:)=(YO2IVCt);
ALLDATAt(3,:)=(rhoTDCt);
ALLDATAt(4,:)=Nt;
ALLDATAt(5,:)=(TTDCt);
ALLDATAt(6,:)=(Twt);
ALLDATAt(7,:)=(SOE1t);
ALLDATAt(8,:)=(FrPulse1t);
ALLDATAt(9,:)=(SOE2t);
ALLDATAt(10,:)=(FrPulse2t);
ALLDATAt(11,:)=(SOE3t);
ALLDATAt(12,:)=(FrPulse3t);
ALLDATAt(13,:)=(SOE4t);
ALLDATAt(14,:)=(FrPulse4t);
%ALLDATAt(15,:)=(PTDCt);
%ALLDATAt(16,:)=(Frt);
%ALLDATAt(17,:)=(IDt);
%ALLDATAt(18,:)=EOCt;


   % .For the cold and hot transients. %

%Cold%
ALLDATAtC(1,:)=(PinjtC);
ALLDATAtC(2,:)=(YO2IVCtC);
ALLDATAtC(3,:)=(rhoTDCtC);
ALLDATAtC(4,:)=NtC;
ALLDATAtC(5,:)=(TTDCtC);
ALLDATAtC(6,:)=(TwtC);
ALLDATAtC(7,:)=(SOE1tC);
ALLDATAtC(8,:)=(FrPulse1tC);
ALLDATAtC(9,:)=(SOE2tC);
ALLDATAtC(10,:)=(FrPulse2tC);
ALLDATAtC(11,:)=(SOE3tC);
ALLDATAtC(12,:)=(FrPulse3tC);
ALLDATAtC(13,:)=(SOE4tC);
ALLDATAtC(14,:)=(FrPulse4tC);
%ALLDATAtC(15,:)=(PTDCtC);
%ALLDATAtC(16,:)=(FrtC);
%ALLDATAtC(17,:)=(IDtC);
%ALLDATAtC(18,:)=EOCtC;


%Hot%
ALLDATAtH(1,:)=(PinjtH);
ALLDATAtH(2,:)=(YO2IVCtH);
ALLDATAtH(3,:)=(rhoTDCtH);
```

```
ALLDATAtH(4,:)=NtH;
ALLDATAtH(5,:)=(TTDCtH);
ALLDATAtH(6,:)=(TwtH);
ALLDATAtH(7,:)=(SOE1tH);
ALLDATAtH(8,:)=(FrPulse1tH);
ALLDATAtH(9,:)=(SOE2tH);
ALLDATAtH(10,:)=(FrPulse2tH);
ALLDATAtH(11,:)=(SOE3tH);
ALLDATAtH(12,:)=(FrPulse3tH);
ALLDATAtH(13,:)=(SOE4tH);
ALLDATAtH(14,:)=(FrPulse4tH);
%ALLDATAtH(15,:)=(PTDCtH);
%ALLDATAtH(16,:)=(FrtH);
%ALLDATAtH(17,:)=(IDtH);
%ALLDATAtH(18,:)=EOCtH;


      % .. For excluded values (SOOT) .. %

    % .. Steady .. %
ALLDATA2(1,:)=(Pinj(posYSOOT02));
ALLDATA2(2,:)=(YO2IVC(posYSOOT02));
ALLDATA2(3,:)=(rhoTDC(posYSOOT02));
ALLDATA2(4,:)=N(posYSOOT02);
ALLDATA2(5,:)=(TTDC(posYSOOT02));
ALLDATA2(6,:)=(Tw(posYSOOT02));
ALLDATA2(7,:)=(SOE1(posYSOOT02));
ALLDATA2(8,:)=(FrPulse1(posYSOOT02));
ALLDATA2(9,:)=(SOE2(posYSOOT02));
ALLDATA2(10,:)=(FrPulse2(posYSOOT02));
ALLDATA2(11,:)=(SOE3(posYSOOT02));
ALLDATA2(12,:)=(FrPulse3(posYSOOT02));
ALLDATA2(13,:)=(SOE4(posYSOOT02));
ALLDATA2(14,:)=(FrPulse4(posYSOOT02));
%ALLDATA2(15,:)=(PTDC(posYSOOT02)); % Added TDC pressure  (still not used)
%ALLDATA2(15,:)=(Fr(posYSOOT02));
%ALLDATA2(16,:)=(ID);
%ALLDATA2(17,:)=EOC;


    % .. Transient .. %
ALLDATAt2(1,:)=(Pinjt(posYSOOTt02));
ALLDATAt2(2,:)=(YO2IVCt(posYSOOTt02));
ALLDATAt2(3,:)=(rhoTDCt(posYSOOTt02));
ALLDATAt2(4,:)=Nt(posYSOOTt02);
ALLDATAt2(5,:)=(TTDCt(posYSOOTt02));
ALLDATAt2(6,:)=(Twt(posYSOOTt02));
ALLDATAt2(7,:)=(SOE1t(posYSOOTt02));
ALLDATAt2(8,:)=(FrPulse1t(posYSOOTt02));
ALLDATAt2(9,:)=(SOE2t(posYSOOTt02));
ALLDATAt2(10,:)=(FrPulse2t(posYSOOTt02));
ALLDATAt2(11,:)=(SOE3t(posYSOOTt02));
ALLDATAt2(12,:)=(FrPulse3t(posYSOOTt02));
ALLDATAt2(13,:)=(SOE4t(posYSOOTt02));
ALLDATAt2(14,:)=(FrPulse4t(posYSOOTt02));
%ALLDATAt2(15,:)=(PTDCt(posYSOOTt02));
%ALLDATAt2(15,:)=(Frt(posYSOOTt02));
%ALLDATA2(16,:)=(ID);
%ALLDATA2(17,:)=EOC;
```

```
%For the hot and cold transients

%Cold:
ALLDATAt2C(1,:)=(PinjtC(posYSOOTtC02));
ALLDATAt2C(2,:)=(YO2IVCtC(posYSOOTtC02));
ALLDATAt2C(3,:)=(rhoTDCtC(posYSOOTtC02));
ALLDATAt2C(4,:)=NtC(posYSOOTtC02);
ALLDATAt2C(5,:)=(TTDCtC(posYSOOTtC02));
ALLDATAt2C(6,:)=(TwtC(posYSOOTtC02));
ALLDATAt2C(7,:)=(SOE1tC(posYSOOTtC02));
ALLDATAt2C(8,:)=(FrPulse1tC(posYSOOTtC02));
ALLDATAt2C(9,:)=(SOE2tC(posYSOOTtC02));
ALLDATAt2C(10,:)=(FrPulse2tC(posYSOOTtC02));
ALLDATAt2C(11,:)=(SOE3tC(posYSOOTtC02));
ALLDATAt2C(12,:)=(FrPulse3tC(posYSOOTtC02));
ALLDATAt2C(13,:)=(SOE4tC(posYSOOTtC02));
ALLDATAt2C(14,:)=(FrPulse4tC(posYSOOTtC02));
%ALLDATAt2C(15,:)=(PTDCtC(posYSOOTtC02));
%ALLDATAt2(15,:)=(Frt(posYSOOTt02));
%ALLDATA2(16,:)=(ID);
%ALLDATA2(17,:)=EOC;


%Cold:
ALLDATAt2H(1,:)=(PinjtH(posYSOOTtH02));
ALLDATAt2H(2,:)=(YO2IVCtH(posYSOOTtH02));
ALLDATAt2H(3,:)=(rhoTDCtH(posYSOOTtH02));
ALLDATAt2H(4,:)=NtH(posYSOOTtH02);
ALLDATAt2H(5,:)=(TTDCtH(posYSOOTtH02));
ALLDATAt2H(6,:)=(TwtH(posYSOOTtH02));
ALLDATAt2H(7,:)=(SOE1tH(posYSOOTtH02));
ALLDATAt2H(8,:)=(FrPulse1tH(posYSOOTtH02));
ALLDATAt2H(9,:)=(SOE2tH(posYSOOTtH02));
ALLDATAt2H(10,:)=(FrPulse2tH(posYSOOTtH02));
ALLDATAt2H(11,:)=(SOE3tH(posYSOOTtH02));
ALLDATAt2H(12,:)=(FrPulse3tH(posYSOOTtH02));
ALLDATAt2H(13,:)=(SOE4tH(posYSOOTtH02));
ALLDATAt2H(14,:)=(FrPulse4tH(posYSOOTtH02));
%ALLDATAt2H(15,:)=(PTDCtH(posYSOOTtH02));
%ALLDATAt2(15,:)=(Frt(posYSOOTt02));
%ALLDATA2(16,:)=(ID);
%ALLDATA2(17,:)=EOC;


% Preprocessing of outputs; using log lead us to better predictions

normYCO=abs(log(YCO));
normYHC=abs(log(YHC));
normYSOOT=abs(log(YSOOT));
normYCOt=abs(log(YCOt));
normYHCt=abs(log(YHCt));
normYSOOTt=abs(log(YSOOTt));

%For the new transients
normYCOtC=abs(log(YCOtC));
normYHCtC=abs(log(YHCtC));
normYSOOTtC=abs(log(YSOOTtC));
normYCOtH=abs(log(YCOtH));
normYHCtH=abs(log(YHCtH));
```

```
normYSOOTtH=abs(log(YSOOTtH));
%


%% ## _______ INTRUDUCING A PERTURBATION _______ ## %%

% A perturbation was introduced here on the input data in order to evaluate
% the method's stability
% subscript v is referred to perturbed data

var = 1.03;   %perturbation multiplier

    % .. Steady data variation .. %

clear ALLDATAv;

ALLDATAv(1,:)=(Pinj);
ALLDATAv(2,:)=(YO2IVC/var);%
ALLDATAv(3,:)=(rhoTDC);
ALLDATAv(4,:)=N;
ALLDATAv(5,:)=(TTDC);
ALLDATAv(6,:)=(Tw);
ALLDATAv(7,:)=(SOE1);
ALLDATAv(8,:)=(FrPulse1);
ALLDATAv(9,:)=(SOE2);
ALLDATAv(10,:)=(FrPulse2);
ALLDATAv(11,:)=(SOE3);
ALLDATAv(12,:)=(FrPulse3);
ALLDATAv(13,:)=(SOE4);
ALLDATAv(14,:)=(FrPulse4);
%ALLDATAv(15,:)=(PTDC*10^5);
%ALLDATAv(16,:)=(Fr);
%ALLDATAv(17,:)=(ID);
%ALLDATAv(18,:)=EOC;



    % .. Transient data variation .. %

%for the ambient data
clear ALLDATAtv;

ALLDATAtv(1,:)=(Pinjt);
ALLDATAtv(2,:)=(YO2IVCt/var);
ALLDATAtv(3,:)=(rhoTDCt);
ALLDATAtv(4,:)=Nt;
ALLDATAtv(5,:)=(TTDCt);
ALLDATAtv(6,:)=(Twt);
ALLDATAtv(7,:)=(SOE1t);
ALLDATAtv(8,:)=(FrPulse1t);
ALLDATAtv(9,:)=(SOE2t);
ALLDATAtv(10,:)=(FrPulse2t);
ALLDATAtv(11,:)=(SOE3t);
ALLDATAtv(12,:)=(FrPulse3t);
ALLDATAtv(13,:)=(SOE4t);
ALLDATAtv(14,:)=(FrPulse4t);
%ALLDATAtv(15,:)=(PTDCt);
%ALLDATAtv(16,:)=(Frt);
%ALLDATAtv(17,:)=(IDt);
%ALLDATAtv(18,:)=EOCt;
```

```
%the cold data
clear ALLDATAtCv;

ALLDATAtCv(1,:)=(PinjtC);
ALLDATAtCv(2,:)=(YO2IVCtC/var);
ALLDATAtCv(3,:)=(rhoTDCtC);
ALLDATAtCv(4,:)=NtC;
ALLDATAtCv(5,:)=(TTDCtC);
ALLDATAtCv(6,:)=(TwtC);
ALLDATAtCv(7,:)=(SOE1tC);
ALLDATAtCv(8,:)=(FrPulse1tC);
ALLDATAtCv(9,:)=(SOE2tC);
ALLDATAtCv(10,:)=(FrPulse2tC);
ALLDATAtCv(11,:)=(SOE3tC);
ALLDATAtCv(12,:)=(FrPulse3tC);
ALLDATAtCv(13,:)=(SOE4tC);
ALLDATAtCv(14,:)=(FrPulse4tC);
%ALLDATAtCv(15,:)=(PTDCtC);
%ALLDATAtCv(16,:)=(FrtC);
%ALLDATAtCv(17,:)=(IDtC);
%ALLDATAtCv(18,:)=EOCtC;


%and the hot data
clear ALLDATAtHv;

ALLDATAtHv(1,:)=(PinjtH);
ALLDATAtHv(2,:)=(YO2IVCtH/var);
ALLDATAtHv(3,:)=(rhoTDCtH);
ALLDATAtHv(4,:)=NtH;
ALLDATAtHv(5,:)=(TTDCtH);
ALLDATAtHv(6,:)=(TwtH);
ALLDATAtHv(7,:)=(SOE1tH);
ALLDATAtHv(8,:)=(FrPulse1tH);
ALLDATAtHv(9,:)=(SOE2tH);
ALLDATAtHv(10,:)=(FrPulse2tH);
ALLDATAtHv(11,:)=(SOE3tH);
ALLDATAtHv(12,:)=(FrPulse3tH);
ALLDATAtHv(13,:)=(SOE4tH);
ALLDATAtHv(14,:)=(FrPulse4tH);
%ALLDATAtHv(15,:)=(PTDCtH);
%ALLDATAtHv(16,:)=(FrtH);
%ALLDATAtHv(17,:)=(IDtH);
%ALLDATAtHv(18,:)=EOCtH;



    % .. For excluded values (SOOT) .. %

    % .. Steady .. %
clear ALLDATA2v;

ALLDATA2v(1,:)=(Pinj(posYSOOT02));
ALLDATA2v(2,:)=(YO2IVC(posYSOOT02)/var);
ALLDATA2v(3,:)=(rhoTDC(posYSOOT02));
ALLDATA2v(4,:)=N(posYSOOT02);
ALLDATA2v(5,:)=(TTDC(posYSOOT02));
ALLDATA2v(6,:)=(Tw(posYSOOT02));
```

```
ALLDATA2v(7,:)=(SOE1(posYSOOT02));
ALLDATA2v(8,:)=(FrPulse1(posYSOOT02));
ALLDATA2v(9,:)=(SOE2(posYSOOT02));
ALLDATA2v(10,:)=(FrPulse2(posYSOOT02));
ALLDATA2v(11,:)=(SOE3(posYSOOT02));
ALLDATA2v(12,:)=(FrPulse3(posYSOOT02));
ALLDATA2v(13,:)=(SOE4(posYSOOT02));
ALLDATA2v(14,:)=(FrPulse4(posYSOOT02));
%ALLDATA2v(15,:)=(PTDC(posYSOOT02));
%ALLDATA2v(15,:)=(Fr(posYSOOT02));
%ALLDATA2v(16,:)=(ID);
%ALLDATA2v(17,:)=EOC;



    % .. Transient .. %

%for the ambient dataa

clear ALLDATAt2v;

ALLDATAt2v(1,:)=(Pinjt(posYSOOTt02));
ALLDATAt2v(2,:)=(YO2IVCt(posYSOOTt02)/var);
ALLDATAt2v(3,:)=(rhoTDCt(posYSOOTt02));
ALLDATAt2v(4,:)=Nt(posYSOOTt02);
ALLDATAt2v(5,:)=(TTDCt(posYSOOTt02));
ALLDATAt2v(6,:)=(Twt(posYSOOTt02));
ALLDATAt2v(7,:)=(SOE1t(posYSOOTt02));
ALLDATAt2v(8,:)=(FrPulse1t(posYSOOTt02));
ALLDATAt2v(9,:)=(SOE2t(posYSOOTt02));
ALLDATAt2v(10,:)=(FrPulse2t(posYSOOTt02));
ALLDATAt2v(11,:)=(SOE3t(posYSOOTt02));
ALLDATAt2v(12,:)=(FrPulse3t(posYSOOTt02));
ALLDATAt2v(13,:)=(SOE4t(posYSOOTt02));
ALLDATAt2v(14,:)=(FrPulse4t(posYSOOTt02));
%ALLDATAt2v(15,:)=(PTDCt(posYSOOTt02));
%ALLDATAt2v(15,:)=(Frt(posYSOOTt02));
%ALLDATA2v(16,:)=(ID);
%ALLDATA2v(17,:)=EOC;

%the cold cata

clear ALLDATAt2Cv

ALLDATAt2Cv(1,:)=(PinjtC(posYSOOTtC02));
ALLDATAt2Cv(2,:)=(YO2IVCtC(posYSOOTtC02)/var);
ALLDATAt2Cv(3,:)=(rhoTDCtC(posYSOOTtC02));
ALLDATAt2Cv(4,:)=NtC(posYSOOTtC02);
ALLDATAt2Cv(5,:)=(TTDCtC(posYSOOTtC02));
ALLDATAt2Cv(6,:)=(TwtC(posYSOOTtC02));
ALLDATAt2Cv(7,:)=(SOE1tC(posYSOOTtC02));
ALLDATAt2Cv(8,:)=(FrPulse1tC(posYSOOTtC02));
ALLDATAt2Cv(9,:)=(SOE2tC(posYSOOTtC02));
ALLDATAt2Cv(10,:)=(FrPulse2tC(posYSOOTtC02));
ALLDATAt2Cv(11,:)=(SOE3tC(posYSOOTtC02));
ALLDATAt2Cv(12,:)=(FrPulse3tC(posYSOOTtC02));
ALLDATAt2Cv(13,:)=(SOE4tC(posYSOOTtC02));
ALLDATAt2Cv(14,:)=(FrPulse4tC(posYSOOTtC02));
%ALLDATAt2Cv(15,:)=(PTDCtC(posYSOOTtC02));
%ALLDATAt2v(15,:)=(Frt(posYSOOTt02));
```

```
%ALLDATA2v(16,:)=(ID);
%ALLDATA2v(17,:)=EOC;

%and the hot data

clear ALLDATAt2Hv

ALLDATAt2Hv(1,:)=(PinjtH(posYSOOTtH02));
ALLDATAt2Hv(2,:)=(YO2IVCtH(posYSOOTtH02)/var);
ALLDATAt2Hv(3,:)=(rhoTDCtH(posYSOOTtH02));
ALLDATAt2Hv(4,:)=NtH(posYSOOTtH02);
ALLDATAt2Hv(5,:)=(TTDCtH(posYSOOTtH02));
ALLDATAt2Hv(6,:)=(TwtH(posYSOOTtH02));
ALLDATAt2Hv(7,:)=(SOE1tH(posYSOOTtH02));
ALLDATAt2Hv(8,:)=(FrPulse1tH(posYSOOTtH02));
ALLDATAt2Hv(9,:)=(SOE2tH(posYSOOTtH02));
ALLDATAt2Hv(10,:)=(FrPulse2tH(posYSOOTtH02));
ALLDATAt2Hv(11,:)=(SOE3tH(posYSOOTtH02));
ALLDATAt2Hv(12,:)=(FrPulse3tH(posYSOOTtH02));
ALLDATAt2Hv(13,:)=(SOE4tH(posYSOOTtH02));
ALLDATAt2Hv(14,:)=(FrPulse4tH(posYSOOTtH02));
%ALLDATAt2Hv(15,:)=(PTDCtH(posYSOOTtH02));
%ALLDATAt2v(15,:)=(Frt(posYSOOTt02));
%ALLDATA2v(16,:)=(ID);
%ALLDATA2v(17,:)=EOC;



%% Set a training base
    % Training set creation for each pollutant. Some data was removed since
    % it brought errors to predictions.

clear TRAIN; clear TRAINSOOT; clear TRAINOUTHC; clear TRAINOUTCO; clear TRAINOUTSOOT;

clear TRAIN; clear TRAINSOOT; clear TRAINOUTHC; clear TRAINOUTCO; clear TRAINOUTSOOT;

TRAIN=[ALLDATA ALLDATAt ALLDATAtC ALLDATAtH]; %Introducing the new 2 transients
TRAINSOOT=[ALLDATA2 ALLDATAt2 ALLDATAt2C ALLDATAt2H];
TRAINOUTHC=[normYHC ; normYHCt ; normYHCtC ; normYHCtH]; %cold and hot outputs added
TRAINOUTCO=[normYCO ; normYCOt ; normYCOtC ; normYCOtH]; %cold and hot outputs added
TRAINOUTSOOT=[normYSOOT ; normYSOOTt ; normYSOOTtC ; normYSOOTtH]; %cold and hot↙
outputs added


%% Set a training base for the perturbed data

clear TRAINv; clear TRAINv2;

TRAINv=[ALLDATAv ALLDATAtv ALLDATAtCv ALLDATAtHv];
TRAINv2=[ALLDATA2v ALLDATAt2v ALLDATAt2Cv ALLDATAt2Hv];



%% ------------------------------------------------ HC Network↙
----------------------------------------

networkHC=[ 20 ];   %[n n n] creates a ANN with 3 hidden-layers and n nodes each%
parameter_number= 14*networkHC(1,1); %for 14 imputs with 2 layers, change if needed,↙
it only displays a number so you don't have to calculate it manually
best_vperf = 100;  ii = 0; best_vperf_out = zeros((training_iterations),1); %↙
```

```
preallocating a "fake values" to initate the iterative process


%_______ Searching the error limit ________%

%inicializating variables that will be used:
counter=1;
Error_HC=[];
R2HC=[];
counter_pre=1;

    while ii < training_iterations

        clear netStatHC
netStatHC=feedforwardnet([networkHC]);  %set the type of ANN "feedforwardnet"%
netStatHC.trainFcn = 'trainlm';

for i=1:size(networkHC,2)
    netStatHC.layers{i}.transferFcn='logsig'; % 'logsig' is a transfer function.↙
Transfer functions calculate a layer's output from its net input%
end
%netStatHC.layers{2}.transferFcn='logsig';
netStatHC.trainParam.epochs=800;        % Maximum number of iterations
netStatHC.trainParam.min_grad=1E-56;    % Very small, to achieve the best result
netStatHC.trainParam.max_fail=60;       % Number of iterations after the minimum (to↙
check if the error reduces or not)
netStatHC.trainParam.show=35;           % It's something related to showing the↙
results, but has an impact on the training process
netStatHC.divideparam.trainratio=0.70;  % Train set
netStatHC.divideparam.valratio=0.15;    % Validation set
netStatHC.divideparam.testratio=0.15;   % Testing set
netStatHC.trainParam.lr = 0.007;        % Learning Rate (it is, somehow, a calculation↙
step)


[netStatHC,tr]=train(netStatHC,TRAIN,transpose(TRAINOUTHC));

      %Getting the randomly created test and training bases set by Matlab

clear TESTHC; clear TESTOUTHC; clear REALTRAINHC; clear OUTTRAINHC

for i=1:size(tr.testInd,2)
     TESTHC(:,i)=TRAIN(:,tr.testInd(i));
     TESTOUTHC(i,1)=TRAINOUTHC(tr.testInd(i),1);
end
for i=1:size(tr.trainInd,2)
    REALTRAINHC(:,i)=TRAIN(:,tr.trainInd(i));
    OUTTRAINHC(i,1)=TRAINOUTHC(tr.trainInd(i),1);
end
    best_vperf = tr.best_vperf;
    best_vperf_out(ii+1) = best_vperf;
    ii = ii + 1;

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % Since the first iterations to get an acceptable error could be useful,
 % we also evaluate their stability and R^2 because one of them could even
 % be better than the following Functions created (explained a bit later)

 %We evaluate the recently trained network with the data imputs:%
```

```
  yHC2=netStatHC(TRAIN);    % sub 2 only to differ different data
  yHC=exp(-yHC2);      % yHC are the predicted values based only on inputs
  yHC_exp=exp(-TRAINOUTHC);


  yHCvar2=netStatHC(TRAINv);    % sub 2 only to differ differents datas
  yHCvar=exp(-yHCvar2);           % sub var is related to perturbed data


  %%% ______ COMPARISON _______ %%%(real vs predicted and predicted vs perturbed)


[~,sizeHC] = size(yHC);        [~,sizeHCvar] = size(yHCvar);


clear rel_errorHC  ; clear rel_errorCO ; clear rel_errorSOOT
clear rel_errorHCvar ; clear rel_errorCOvar; clear rel_errorSOOTvar
rel_errorHC_pre = zeros(sizeHC,1); %preallocating to increse the speed code%
rel_errorHC2_pre = zeros(sizeHCvar,1);

% Relative error - original data
for i=1:sizeHC
        rel_errorHC_pre(i) = ((yHC(i) - yHC_exp(i))/yHC_exp(i)) * 100;
end


% Variation - perturbed/predicted data
for l=1:sizeHCvar
        rel_errorHC2_pre(l) = ((yHCvar(l) - yHC(l))/yHC(l)) * 100;
end


% Mean of relative error

zero = zeros(sizeHC,1);

mean_rel_errorHC_pre = mean(abs(rel_errorHC_pre)) + zero;

mean_rel_errorHC2_pre = mean(abs(rel_errorHC2_pre)) + zero;


% Estimating the Coefficient of Determination between predicted and predicited +↙
variation

HC = fitlm(yHC,yHCvar);
r_srq_hc_pre = HC.Rsquared.Ordinary;


Error_HC_pre(1,counter_pre)=mean_rel_errorHC2_pre(1,1);
R2HC_pre(1,counter_pre)=r_srq_hc_pre;

Error_HC_pre(2,counter_pre)=counter_pre;
R2HC_pre(2,counter_pre)=counter_pre;


%normalizing the error:
Error_HC_log_pre = log(Error_HC_pre(1,:));
std_dev_HC_pre = std(Error_HC_log_pre);
avg_HC_pre = mean(Error_HC_log_pre);
error_limit_hc_relative_log_pre = avg_HC_pre-2*std_dev_HC_pre;
```

```
error_limit_hc_relative_pre=exp(error_limit_hc_relative_log_pre);


%Generating the Network function:

title_=['HC_pre',num2str(counter_pre)];
genFunction(netStatHC, title_ , 'MatrixOnly', 'yes');


%%%The weights and biases are calculated for each function created with each loop↙
iteration%%%

%HC weights and biases.%

 for i=1:size(networkHC,2)+1
    if i==1
        temp=netStatHC.IW{i};
        WHC_pre{i,counter_pre}=reformMat(temp);
        bHC_pre{i,counter_pre}=transpose(netStatHC.b{i});
    else
        temp=netStatHC.LW{i+(i-2)*(size(networkHC,2)+1)};
        WHC_pre{i,counter_pre}=reformMat(temp);
        bHC_pre{i,counter_pre}=transpose(netStatHC.b{i});
    end
 end


  counter_pre=counter_pre+1;

    end


best_vperf_out_log = log(best_vperf_out);
std_dev = std(best_vperf_out_log);
avg = mean(best_vperf_out_log);
error_limit_hc = exp((Z_target * std_dev) + avg); %This error is the Mean Squared↙
Error (MSE)of validation process of training, based on normal distribution



%-------------------------------%
%searching for the best network stats (to get track of the best r^2 and error):

Best_network_err_HC_pre=min(Error_HC_pre(1,:));
Best_network_R2_HC_pre=max(R2HC_pre(1,:));

iii=1;

[lineHC , rowHC]=size(Error_HC_pre);

%So it doesn't return an error in case only one function has been generated.
%(optional in this part, but necessary later)
 while iii<=length(Error_HC_pre) && rowHC>1

 if Best_network_err_HC_pre == Error_HC_pre(1,iii)
     bestHCerr_pre=Error_HC_pre(:,iii);
 end
 if Best_network_R2_HC_pre == R2HC_pre(1,iii)
     bestHCR2_pre=R2HC_pre(:,iii);
 end
```

```
    iii=iii+1;
 end

if rowHC == 1

    bestHCerr_pre=Error_HC_pre(:,1);
    bestHCR2_pre=R2HC_pre(:,1);
end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%_______ Training a new Neural Network to achieve the error previosly determined ________%
 error_hc2=10;   error_limit_hc_relative=100; % introducing and prealocating the new errors
 error_hc = 10; %preallocating a "fake error" to initate the iterative process
while (error_hc > error_limit_hc || error_limit_hc_relative > error_hc2)  && counter <= training_iterations

        clear netStatHC
netStatHC=feedforwardnet([networkHC]);  %set the type of ANN "feedforwardnet"%
netStatHC.trainFcn = 'trainlm';

for i=1:size(networkHC,2)
    netStatHC.layers{i}.transferFcn='logsig'; % 'logsig' is a transfer function. Transfer functions calculate a layer’s output from its net input%
end
%netStatHC.layers{2}.transferFcn='logsig';
netStatHC.trainParam.epochs=800;          % Maximum number of iterations
netStatHC.trainParam.min_grad=1E-56;      % Very small, to achieve the best result
netStatHC.trainParam.max_fail=60;         % Number of iterations after the minimum (to check if the error reduces or not)
netStatHC.trainParam.show=35;             % It's something related to showing the results, but has an impact on the training process
netStatHC.divideparam.trainratio=0.70;  % Train set
netStatHC.divideparam.valratio=0.15;    % Validation set
netStatHC.divideparam.testratio=0.15;   % Testing set
netStatHC.trainParam.lr = 0.007;          % Learning Rate (it is, somehow, a calculation step)

[netStatHC,tr]=train(netStatHC,TRAIN,transpose(TRAINOUTHC));

clear TESTHC; clear TESTOUTHC; clear REALTRAINHC; clear OUTTRAINHC

for i=1:size(tr.testInd,2)
     TESTHC(:,i)=TRAIN(:,tr.testInd(i)); % indicated witch column is related to the best fit %
     TESTOUTHC(i,1)=TRAINOUTHC(tr.testInd(i),1);
end

for i=1:size(tr.trainInd,2)
    REALTRAINHC(:,i)=TRAIN(:,tr.trainInd(i));
    OUTTRAINHC(i,1)=TRAINOUTHC(tr.trainInd(i),1);
end

error_hc = tr.best_vperf;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% At first the code that evaluated the network stability was in another   %
% script, but has been decided that the stability is as important as the  %
% prediction, so both codes have been combined and now with each training %
% loop, everything is evaluated and a function is created, so after the   %
% network has finished all the training, we can decide manually which     %
% funtion is the best and why                                             %


%We evaluate the recently trained network with the data imputs:%

  yHC2=netStatHC(TRAIN);   % sub 2 only to differ different data
  yHC=exp(-yHC2);     % yHC are the predicted values based only on inputs
  yHC_exp=exp(-TRAINOUTHC);


  yHCvar2=netStatHC(TRAINv);   % sub 2 only to differ differents datas
  yHCvar=exp(-yHCvar2);          % sub var is related to perturbed data



 %%% ______ COMPARISON _______ %%% (real vs predicted and predicted vs perturbed)

[~,sizeHC] = size(yHC);       [~,sizeHCvar] = size(yHCvar);


clear rel_errorHC  ; clear rel_errorCO ; clear rel_errorSOOT
clear rel_errorHCvar ; clear rel_errorCOvar; clear rel_errorSOOTvar
rel_errorHC = zeros(sizeHC,1); %preallocating to increse the speed code%
rel_errorHC2 = zeros(sizeHCvar,1);

% Relative error - original data
for i=1:sizeHC
        rel_errorHC(i) = ((yHC(i) - yHC_exp(i))/yHC_exp(i)) * 100;
end


% Variation - perturbed/predicted data
for l=1:sizeHCvar
        rel_errorHC2(l) = ((yHCvar(l) - yHC(l))/yHC(l)) * 100;
end


% Mean of relative error

zero = zeros(sizeHC,1);

mean_rel_errorHC = mean(abs(rel_errorHC)) + zero;

mean_rel_errorHC2 = mean(abs(rel_errorHC2)) + zero;


% Estimating the Coefficient of Determination  between predicted and predicited +↙
variation

HC = fitlm(yHC,yHCvar);
r_srq_hc = HC.Rsquared.Ordinary;


Error_HC(1,counter)=mean_rel_errorHC2(1,1);
```

```
R2HC(1,counter)=r_srq_hc;

Error_HC(2,counter)=counter;
R2HC(2,counter)=counter;


%normalizing the error:
Error_HC_log = log(Error_HC(1,:));
std_dev_HC = std(Error_HC_log);
avg_HC = mean(Error_HC_log);
error_limit_hc_relative_log = avg_HC-2*std_dev_HC;
error_limit_hc_relative=exp(error_limit_hc_relative_log);


%Generating the Network function:

title_=['HC',num2str(counter)];
genFunction(netStatHC, title_ , 'MatrixOnly', 'yes');



%Saving the current HC network weights and biases.

 for i=1:size(networkHC,2)+1
    if i==1
        temp=netStatHC.IW{i};
        WHC{i,counter}=reformMat(temp);
        bHC{i,counter}=transpose(netStatHC.b{i});
    else
        temp=netStatHC.LW{i+(i-2)*(size(networkHC,2)+1)};
        WHC{i,counter}=reformMat(temp);
        bHC{i,counter}=transpose(netStatHC.b{i});
    end
 end

counter=counter+1;

end

%%%Ranges and ohter data needed to export the network%%%

%Ranges used to normalize Inputs

for i=1:size(ALLDATA,1)

    minOUTHC(i,1)=min(TRAIN(i,:));
    maxOUTHC(i,1)=max(TRAIN(i,:));

end

%Ranges used to denormalize Outputs (NOT AUTOMATISED)

minmaxHC(1,1)=min(TRAINOUTHC);
minmaxHC(2,1)=max(TRAINOUTHC);

%Networks 'layers' number of nodes
nbNodesHC=[ networkHC 1];
```

```
%------------------------------%
%searching for the best network stats:

Best_network_err_HC=min(Error_HC(1,:));
Best_network_R2_HC=max(R2HC(1,:));

iii=1;

[lineHC , rowHC]=size(Error_HC);%So it doesn't return an error in case only one↙
function has been generated.


 while iii<=length(Error_HC) && rowHC>1

 if Best_network_err_HC == Error_HC(1,iii)
     bestHCerr=Error_HC(:,iii);
 end
 if Best_network_R2_HC == R2HC(1,iii)
     bestHCR2=R2HC(:,iii);
 end
    iii=iii+1;
 end

if rowHC == 1

    bestHCerr=Error_HC(:,1);
    bestHCR2=R2HC(:,1);
end




%% ---------------------------------------- SOOT Network↙
-----------------------------------------
    %Creation and training of SOOT network

 networkSOOT=[ 20 ];   %[n n n] creat a ANN with 3 hidden-layers and n nodes each%

best_vperf = 100;  ii = 0; best_vperf_out = zeros((training_iterations),1); %↙
preallocating a "fake values" to initate the iterative process
clear counter counter_pre;
counter= 1; counter_pre=1; %initializing variables used in the loop
Error_SOOT=[];
R2SOOT=[];


%_______ Searching the error limit ________%

    while ii < training_iterations

clear netStatSOOT
netStatSOOT=feedforwardnet([networkSOOT]);
netStatSOOT.trainFcn = 'trainlm';

for i=1:size(networkSOOT,2)
    netStatSOOT.layers{i}.transferFcn='logsig';
end
%netStatSOOT.layers{1}.transferFcn='logsig';
%netStatSOOT.layers{2}.transferFcn='logsig';
```

```
netStatSOOT.trainParam.epochs=800;
netStatSOOT.trainParam.min_grad=1E-56;
netStatSOOT.trainParam.max_fail=60;
netStatSOOT.trainParam.show=5;
netStatSOOT.divideparam.trainratio=0.70;
netStatSOOT.divideparam.valratio=0.15;
netStatSOOT.divideparam.testratio=0.15;
netStatSOOT.trainParam.lr = 0.05;
[netStatSOOT,tr]=train(netStatSOOT,TRAINSOOT,transpose(TRAINOUTSOOT));

clear TESTSOOT; clear TESTOUTSOOT; clear REALTRAINSOOT; clear OUTTRAINSOOT

for i=1:size(tr.testInd,2)
     TESTSOOT(:,i)=TRAINSOOT(:,tr.testInd(i));
     TESTOUTSOOT(i,1)=TRAINOUTSOOT(tr.testInd(i),1);
end

for i=1:size(tr.trainInd,2)
    REALTRAINSOOT(:,i)=TRAINSOOT(:,tr.trainInd(i));
    OUTTRAINSOOT(i,1)=TRAINOUTSOOT(tr.trainInd(i),1);
end

    best_vperf = tr.best_vperf;
    best_vperf_out(ii+1) = best_vperf;
    ii = ii + 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


 %We evaluate the recently trained network with the data imputs:%

  ySOOT2=netStatSOOT(TRAIN);   % sub 2 only to differ different data
  ySOOT=exp(-ySOOT2);     % yHC are the predicted values based only on inputs
  ySOOT_exp=exp(-TRAINOUTSOOT);


  ySOOTvar2=netStatSOOT(TRAINv);   % sub 2 only to differ differents datas
  ySOOTvar=exp(-ySOOTvar2);          % sub var is related to perturbed data


  % ______ COMPARISON _______ %%

[~,sizeSOOT] = size(ySOOT);       [~,sizeSOOTvar] = size(ySOOTvar);


clear rel_errorHC  ; clear rel_errorCO ; clear rel_errorSOOT
clear rel_errorHCvar ; clear rel_errorCOvar; clear rel_errorSOOTvar
rel_errorSOOT_pre = zeros(sizeSOOT,1); %preallocating to increse the speed code%
rel_errorSOOT2_pre = zeros(sizeSOOTvar,1);


% Relative error - original data

%for k=1:sizeSOOT
%        rel_errorSOOT_pre(k) = ((ySOOT(k) - ySOOT_exp(k))/ySOOT_exp(k)) * 100;
%end


% Variation - perturbed/predicted data
```

```
for n=1:sizeSOOTvar
        rel_errorSOOT2_pre(n) = ((ySOOTvar(n) - ySOOT(n))/ySOOT(n)) * 100;
end


% Mean of relative error

zero = zeros(sizeSOOT,1);

mean_rel_errorSOOT_pre = mean(abs(rel_errorSOOT_pre)) + zero;

mean_rel_errorSOOT2_pre = mean(abs(rel_errorSOOT2_pre)) + zero;



% Estimating the Coefficient of Determination  between predicted and predicited +↙
variation

SOOT = fitlm(ySOOT,ySOOTvar);
r_srq_soot_pre = SOOT.Rsquared.Ordinary;


Error_SOOT_pre(1,counter_pre)=mean_rel_errorSOOT2_pre(1,1);
R2SOOT_pre(1,counter_pre)=r_srq_soot_pre;

Error_SOOT_pre(2,counter_pre)=counter_pre;
R2SOOT_pre(2,counter_pre)=counter_pre;


%normalizing the error:
Error_SOOT_log_pre = log(Error_SOOT_pre(1,:));
std_dev_SOOT_pre = std(Error_SOOT_log_pre);
avg_SOOT_pre = mean(Error_SOOT_log_pre);
error_limit_soot_relative_log_pre = avg_SOOT_pre-2*std_dev_SOOT_pre;
error_limit_soot_relative_pre=exp(error_limit_soot_relative_log_pre);


%Generating the Network function:

title_=['SOOT_pre',num2str(counter_pre)];
genFunction(netStatSOOT, title_ , 'MatrixOnly', 'yes');




%SOOT weights and biases.

 for i=1:size(networkSOOT,2)+1
    if i==1
        temp=netStatSOOT.IW{i};
        WSOOT_pre{i,counter_pre}=reformMat(temp);
        bSOOT_pre{i,counter_pre}=transpose(netStatSOOT.b{i});
    else
        temp=netStatHC.LW{i+(i-2)*(size(networkSOOT,2)+1)};
        WSOOT_pre{i,counter_pre}=reformMat(temp);
        bSOOT_pre{i,counter_pre}=transpose(netStatSOOT.b{i});
    end
 end
```

```
  counter_pre=counter_pre+1;

end

best_vperf_out_log = log(best_vperf_out);
std_dev = std(best_vperf_out_log);
avg = mean(best_vperf_out_log);
error_limit_soot = exp((Z_target * std_dev) + avg); %This error is the Mean Squared↙
Error (MSE)of validation process of training







%_______ Training a new Neural Network to achieve the error previosly determined↙
________%

error_soot2=10;   error_limit_soot_relative=100; % introducing and prealocating the↙
new errors
 error_soot = 10; %preallocating a "fake error" to initate the iterative process
while (error_soot > error_limit_soot || error_limit_soot_relative > error_soot2)  &&↙
counter <= training_iterations

       clear netStatSOOT
netStatSOOT=feedforwardnet([networkSOOT]);
netStatSOOT.trainFcn = 'trainlm';

for i=1:size(networkSOOT,2)
    netStatSOOT.layers{i}.transferFcn='logsig';
end
netStatSOOT.trainParam.epochs=800;
netStatSOOT.trainParam.min_grad=1E-56;
netStatSOOT.trainParam.max_fail=60;
netStatSOOT.trainParam.show=5;
netStatSOOT.divideparam.trainratio=0.70;
netStatSOOT.divideparam.valratio=0.15;
netStatSOOT.divideparam.testratio=0.15;
netStatSOOT.trainParam.lr = 0.05;
[netStatSOOT,tr]=train(netStatSOOT,TRAINSOOT,transpose(TRAINOUTSOOT));

clear TESTSOOT; clear TESTOUTSOOT; clear REALTRAINSOOT; clear OUTTRAINSOOT

for i=1:size(tr.testInd,2)
     TESTSOOT(:,i)=TRAINSOOT(:,tr.testInd(i));
     TESTOUTSOOT(i,1)=TRAINOUTSOOT(tr.testInd(i),1);
end

for i=1:size(tr.trainInd,2)
    REALTRAINSOOT(:,i)=TRAINSOOT(:,tr.trainInd(i));
    OUTTRAINSOOT(i,1)=TRAINOUTSOOT(tr.trainInd(i),1);
end

error_soot = tr.best_vperf;



%We evaluate the recently trained network with the data imputs:
ySOOT2=netStatSOOT(TRAINSOOT);
```

```
ySOOT=exp(-ySOOT2);
ySOOT_exp=exp(-TRAINOUTSOOT);


ySOOTvar2=netStatSOOT(TRAINv2);
ySOOTvar=exp(-ySOOTvar2);


% ______ COMPARISON _______ %


[~,sizeSOOT] = size(ySOOT);    [~,sizeSOOTvar] = size(ySOOTvar);

clear rel_errorHC  ; clear rel_errorCO ; clear rel_errorSOOT
clear rel_errorHCvar ; clear rel_errorCOvar; clear rel_errorSOOTvar
rel_errorSOOT = zeros(sizeSOOT,1);  %preallocating to increse the speed code%
rel_errorSOOT2 = zeros(sizeSOOTvar,1);

% Relative error - original data


for k=1:sizeSOOT
        rel_errorSOOT(k) = ((ySOOT(k) - ySOOT_exp(k))/ySOOT_exp(k)) * 100;
end

% Variation - perturbed/predicted data


for n=1:sizeSOOTvar
        rel_errorSOOT2(n) = ((ySOOTvar(n) - ySOOT(n))/ySOOT(n)) * 100;
end

% Mean of relative error


zerosoot = zeros(sizeSOOT,1);

mean_rel_errorSOOT = mean(abs(rel_errorSOOT)) + zerosoot;

mean_rel_errorSOOT2 = mean(abs(rel_errorSOOT2)) + zerosoot;



%Estimating the Coefficient of Determination  between predicted and predicited +↙
variation


SOOT = fitlm(ySOOT,ySOOTvar);
r_srq_soot = SOOT.Rsquared.Ordinary;


Error_SOOT(1,counter)=mean_rel_errorSOOT2(1,1);
R2SOOT(1,counter)=r_srq_soot;

Error_SOOT(2,counter)=counter;
R2SOOT(2,counter)=counter;


%normalizing the error:
Error_SOOT_log = log(Error_SOOT(1,:));
```

```
std_dev_SOOT = std(Error_SOOT_log);
avg_SOOT = mean(Error_SOOT_log);
error_limit_soot_relative_log = avg_SOOT-2*std_dev_SOOT;
error_limit_soot_relative=exp(error_limit_soot_relative_log);


%Generating the Network function:

title_=['SOOT',num2str(counter)];
genFunction(netStatSOOT, title_ , 'MatrixOnly', 'yes');



% SOOT weights and baises.

for i=1:size(networkSOOT,2)+1
    if i==1
        temp=netStatSOOT.IW{i};
        WSOOT{i,counter}=reformMat(temp);
        bSOOT{i,counter}=transpose(netStatSOOT.b{i});
    else
        temp=netStatSOOT.LW{i+(i-2)*(size(networkSOOT,2)+1)};
        WSOOT{i,counter}=reformMat(temp);
        bSOOT{i,counter}=transpose(netStatSOOT.b{i});
    end
end

 counter=counter+1;

end


%%%Ranges and ohter data needed to export the network%%%

%Ranges used to normalize Inputs
for i=1:size(ALLDATA,1)

    minOUTSOOT(i,1)=min(TRAINSOOT(i,:));
    maxOUTSOOT(i,1)=max(TRAINSOOT(i,:));

end

 %Ranges used to denormalize Outputs (NOT AUTOMATISED)

minmaxSOOT(1,1)=min(TRAINOUTSOOT);
minmaxSOOT(2,1)=max(TRAINOUTSOOT);

%Networks 'layers' number of nodes

nbNodesSOOT=[ networkSOOT 1];


%-------------------------------%
%searching for the best network stats:

Best_network_err_SOOT=min(Error_SOOT(1,:));
Best_network_R2_SOOT=max(R2SOOT(1,:));

iii=1;
```

```
[lineSOOT , rowSOOT]=size(Error_SOOT);%So it doesn't return an error in case
% there is only one funtion generated.

while iii<=length(Error_SOOT) && rowSOOT >1

if Best_network_err_SOOT == Error_SOOT(1,iii)
     bestSOOTerr=Error_SOOT(:,iii);
end
if Best_network_R2_SOOT == R2SOOT(1,iii)
    bestSOOTR2=R2SOOT(:,iii);
end
    iii=iii+1;
end

if rowSOOT == 1

    bestSOOTerr=Error_SOOT(:,1);
    bestSOOTR2=R2SOOT(:,1);
end



%% ---------------------------------------------- CO Network↙
----------------------------------------
    % Creation and training of CO network

networkCO=[ 20 ];   %[n n n] creat a ANN with 3 hidden-layers and n nodes each%

best_vperf = 100;  ii = 0; best_vperf_out = zeros((training_iterations),1); %↙
preallocating a "fake values" to initate the iterative process


clear counter counter_pre; %initializing variables used in the loop
counter= 1; counter_pre= 1;
Error_CO=[];
R2CO=[];

%_______ Searching the error limit ________%

    while ii < training_iterations

clear netStatCO
netStatCO=feedforwardnet(networkCO);
netStatCO.trainFcn = 'trainlm';

for i=1:size(networkCO,2)
    netStatCO.layers{i}.transferFcn='logsig';
end
%netStatCO.layers{1}.transferFcn='logsig';
%netStatCO.layers{2}.transferFcn='logsig';

netStatCO.trainParam.epochs=800;
netStatCO.trainParam.min_grad=1E-10;
netStatCO.trainParam.max_fail=60;
netStatCO.trainParam.show=1;
netStatCO.divideparam.trainratio=0.7;
netStatCO.divideparam.valratio=0.15;
netStatCO.divideparam.testratio=0.15;
netStatCO.trainParam.lr = 0.005;
[netStatCO,tr]=train(netStatCO,TRAIN,transpose(TRAINOUTCO));
```

```
clear TESTCO; clear TESTOUTCO; clear REALTRAINCO; clear OUTTRAINCO

for i=1:size(tr.testInd,2)
     TESTCO(:,i)=TRAIN(:,tr.testInd(i));
     TESTOUTCO(i,1)=TRAINOUTCO(tr.testInd(i),1);
end

for i=1:size(tr.trainInd,2)
    REALTRAINCO(:,i)=TRAIN(:,tr.trainInd(i));
    OUTTRAINCO(i,1)=TRAINOUTCO(tr.trainInd(i),1);
end
    best_vperf = tr.best_vperf;
    best_vperf_out(ii+1) = best_vperf;
    ii = ii + 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%We evaluate the recently trained network with the data imputs:
yCO2=netStatCO(TRAIN);
yCO=exp(-yCO2);
yCO_exp=exp(-TRAINOUTCO);


yCOvar2=netStatCO(TRAINv);
yCOvar=exp(-yCOvar2);


% ______ COMPARISON _______ %

[~,sizeCO] = size(yCO);       [~,sizeCOvar] = size(yCOvar);

clear rel_errorHC  ; clear rel_errorCO ; clear rel_errorSOOT
clear rel_errorHCvar ; clear rel_errorCOvar; clear rel_errorSOOTvar
rel_errorCO_pre = zeros(sizeCO,1);  %preallocating to increse the speed code%
rel_errorCO2_pre = zeros(sizeCOvar,1);

% Relative error - original data

for j=1:sizeCO
        rel_errorCO_pre(j) = ((yCO(j) - yCO_exp(j))/yCO_exp(j)) * 100;
end

% Variation - perturbed/predicted data

for m=1:sizeCOvar
        rel_errorCO2_pre(m) = ((yCOvar(m) - yCO(m))/yCO(m)) * 100;
end

% Mean of relative error

zero = zeros(sizeHC,1);

mean_rel_errorCO_pre = mean(abs(rel_errorCO_pre)) + zero;

mean_rel_errorCO2_pre = mean(abs(rel_errorCO2_pre)) + zero;


%Estimating the Coefficient of Determination  between predicted and predicited +↙
```

```
variation

CO = fitlm(yCO,yCOvar);
r_srq_co_pre = CO.Rsquared.Ordinary;

Error_CO_pre(1,counter_pre)=mean_rel_errorCO2_pre(1,1);
R2CO_pre(1,counter_pre)=r_srq_co_pre;

Error_CO_pre(2,counter_pre)=counter_pre;
R2CO_pre(2,counter_pre)=counter_pre;


%normalizar el error:
Error_CO_log_pre = log(Error_CO_pre(1,:));
std_dev_CO_pre = std(Error_CO_log_pre);
avg_CO_pre = mean(Error_CO_log_pre);
error_limit_co_relative_log_pre = avg_CO_pre-2*std_dev_CO_pre;
error_limit_co_relative_pre=exp(error_limit_co_relative_log_pre);


%Generating the Network function:

title_=['CO_pre',num2str(counter_pre)];
genFunction(netStatCO, title_ , 'MatrixOnly', 'yes');

 % CO weights and biases.

for i=1:size(networkCO,2)+1
    if i==1
        temp=netStatCO.IW{i};
        WCO_pre{i,counter_pre}=reformMat(temp);
        bCO_pre{i,counter_pre}=transpose(netStatCO.b{i});
    else
        temp=netStatCO.LW{i+(i-2)*(size(networkCO,2)+1)};
        WCO_pre{i,counter_pre}=reformMat(temp);
        bCO_pre{i,counter_pre}=transpose(netStatCO.b{i});
    end
end

counter_pre=counter_pre+1;

    end

best_vperf_out_log = log(best_vperf_out);
std_dev = std(best_vperf_out_log);
avg = mean(best_vperf_out_log);
error_limit_co = exp((Z_target * std_dev) + avg); %This error is the Mean Squared↙
Error (MSE)of validation process of training




%_______ Training a new Neural Network to achieve the error previosly determined↙
________%

 error_co2=10;   error_limit_co_relative=100; % introducing and prealocating the new↙
errors
 error_co = 10; %preallocating a "fake error" to initate the iterative process
while (error_co > error_limit_co || error_co2 < error_limit_co_relative) && counter <=↙
```

```
training_iterations

        clear netStatCO
netStatCO=feedforwardnet(networkCO);
netStatCO.trainFcn = 'trainlm';

for i=1:size(networkCO,2)
    netStatCO.layers{i}.transferFcn='logsig';
end
%netStatCO.layers{1}.transferFcn='logsig';
%netStatCO.layers{2}.transferFcn='logsig';

netStatCO.trainParam.epochs=800;
netStatCO.trainParam.min_grad=1E-10;
netStatCO.trainParam.max_fail=60;
netStatCO.trainParam.show=1;
netStatCO.divideparam.trainratio=0.7;
netStatCO.divideparam.valratio=0.15;
netStatCO.divideparam.testratio=0.15;
netStatCO.trainParam.lr = 0.005;
[netStatCO,tr]=train(netStatCO,TRAIN,transpose(TRAINOUTCO));

        clear TESTCO; clear TESTOUTCO; clear REALTRAINCO; clear OUTTRAINCO

for i=1:size(tr.testInd,2)
     TESTCO(:,i)=TRAIN(:,tr.testInd(i));
     TESTOUTCO(i,1)=TRAINOUTCO(tr.testInd(i),1);
end

for i=1:size(tr.trainInd,2)
    REALTRAINCO(:,i)=TRAIN(:,tr.trainInd(i));
    OUTTRAINCO(i,1)=TRAINOUTCO(tr.trainInd(i),1);
end

error_co = tr.best_vperf;

%We evaluate the recently trained network with the data imputs:
yCO2=netStatCO(TRAIN);
yCO=exp(-yCO2);
yCO_exp=exp(-TRAINOUTCO);


yCOvar2=netStatCO(TRAINv);
yCOvar=exp(-yCOvar2);


% ______ COMPARISON _______ %

[~,sizeCO] = size(yCO);       [~,sizeCOvar] = size(yCOvar);

clear rel_errorHC  ; clear rel_errorCO ; clear rel_errorSOOT
clear rel_errorHCvar ; clear rel_errorCOvar; clear rel_errorSOOTvar
rel_errorCO = zeros(sizeCO,1);  %preallocating to increse the speed code%
rel_errorCO2 = zeros(sizeCOvar,1);

% Relative error - original data

for j=1:sizeCO
        rel_errorCO(j) = ((yCO(j) - yCO_exp(j))/yCO_exp(j)) * 100;
end
```

```
% Variation - perturbed/predicted data

for m=1:sizeCOvar
        rel_errorCO2(m) = ((yCOvar(m) - yCO(m))/yCO(m)) * 100;
end

% Mean of relative error

zero = zeros(sizeHC,1);

mean_rel_errorCO = mean(abs(rel_errorCO)) + zero;

mean_rel_errorCO2 = mean(abs(rel_errorCO2)) + zero;


%Estimating the Coefficient of Determination  between predicted and predicited +↙
variation

CO = fitlm(yCO,yCOvar);
r_srq_co = CO.Rsquared.Ordinary;

Error_CO(1,counter)=mean_rel_errorCO2(1,1);
R2CO(1,counter)=r_srq_co;

Error_CO(2,counter)=counter;
R2CO(2,counter)=counter;


%normalizing the error:
Error_CO_log = log(Error_CO(1,:));
std_dev_CO = std(Error_CO_log);
avg_CO = mean(Error_CO_log);
error_limit_co_relative_log = avg_CO-2*std_dev_CO;
error_limit_co_relative=exp(error_limit_co_relative_log);


%Generating the Network function:

title_=['CO',num2str(counter)];
genFunction(netStatCO, title_ , 'MatrixOnly', 'yes');

 % CO weights and biases.

for i=1:size(networkCO,2)+1
    if i==1
        temp=netStatCO.IW{i};
        WCO{i,counter}=reformMat(temp);
        bCO{i,counter}=transpose(netStatCO.b{i});
    else
        temp=netStatCO.LW{i+(i-2)*(size(networkCO,2)+1)};
        WCO{i,counter}=reformMat(temp);
        bCO{i,counter}=transpose(netStatCO.b{i});
    end
end

counter=counter+1;
end
```

```
%(for the weight, bias, and ohter data needed for the exportation of the
% network)

%Ranges used to normalize Inputs

 for i=1:size(ALLDATA,1)

    minOUTCO(i,1)=min(TRAIN(i,:));
    maxOUTCO(i,1)=max(TRAIN(i,:));

 end


 %Ranges used to denormalize Outputs (NOT AUTOMATISED)
 minmaxCO(1,1)=min(TRAINOUTCO);
 minmaxCO(2,1)=max(TRAINOUTCO);

 %Networks 'layers' number of nodes
 nbNodesCO=[ networkCO 1];


%------------------------------%
%searching for the best network stats:

Best_network_err_CO=min(Error_CO(1,:));
Best_network_R2_CO=max(R2CO(1,:));

iii=1;

[lineCO , rowCO]=size(Error_CO); %So it doesn't return an error in case
% there is only one funtion generated.

while iii<=length(Error_CO) && rowCO >1

if Best_network_err_CO == Error_CO(1,iii)
     bestCOerr=Error_CO(:,iii);
end
if Best_network_R2_CO == R2CO(1,iii)
    bestCOR2=R2CO(:,iii);
end
    iii=iii+1;
end

if rowCO == 1

    bestCOerr=Error_CO(:,1);
    bestCOR2=R2CO(:,1);
end




%%  IMPPORTANT: Now the best networks will be displayed on the command window

%HC
if bestHCerr(2,1) == bestHCR2(2,1)

disp(['Best HC network: ',['HC',num2str(bestHCerr(2,1))]]);
disp(['R2 for stability -> ', num2str(bestHCR2(1,1))]);
disp(['Mean error for stability -> ', num2str(bestHCerr(1,1))]);
```

```
else if bestHCerr(2,1) ~= bestHCR2(2,1)

        disp(['Best HC networks: ',['HC',num2str(bestHCerr(2,1))],[' and HC',num2str↙
(bestHCR2(2,1))]]);

        disp(['R2 for stability: ', ['HC',num2str(bestHCR2(2,1))] , [' --> ', num2str↙
(bestHCR2(1,1))]]);
        disp(['R2 for stability: ', ['HC',num2str(bestHCerr(2,1))] , [' --> ',num2str↙
(R2HC(1,bestHCerr(2,1)))]]);

        disp(['Mean error for stability: ', ['HC',num2str(bestHCerr(2,1))], [' --> ',↙
num2str(bestHCerr(1,1))]]);
        disp(['Mean error for stability: ', ['HC',num2str(bestHCR2(2,1))] , [' --> ',↙
num2str(Error_HC(1,bestHCR2(2,1)))]]);
    end
end

%SOOT
if bestSOOTerr(2,1) == bestSOOTR2(2,1)

disp(['Best SOOT network: ',['SOOT',num2str(bestSOOTerr(2,1))]]);
disp(['R2 for stability -> ', num2str(bestSOOTR2(1,1))]);
disp(['Mean error for stability -> ', num2str(bestSOOTerr(1,1))]);

else if bestSOOTerr(2,1) ~= bestSOOTR2(2,1)

        disp(['Best SOOT networks: ',['SOOT',num2str(bestSOOTerr(2,1))],[' and SOOT',↙
num2str(bestSOOTR2(2,1))]]);

        disp(['R2 for stability: ', ['SOOT',num2str(bestSOOTR2(2,1))] , [' --> ',↙
num2str(bestSOOTR2(1,1))]]);
        disp(['R2 for stability: ', ['SOOT',num2str(bestSOOTerr(2,1))] , [' --> ',↙
num2str(R2SOOT(1,bestSOOTerr(2,1)))]]);

        disp(['Mean error for stability: ', ['SOOT',num2str(bestSOOTerr(2,1))] , [' --↙
> ', num2str(bestSOOTerr(1,1))]]);
        disp(['Mean error for stability: ', ['SOOT',num2str(bestSOOTR2(2,1))] , [' -->↙
',num2str(Error_SOOT(1,bestSOOTR2(2,1)))]]);
    end
end


%CO
if bestCOerr(2,1) == bestCOR2(2,1)

disp(['Best CO network: ',['CO',num2str(bestCOerr(2,1))]]);
disp(['R2 for stability -> ', num2str(bestCOR2(1,1))]);
disp(['Mean error for stability -> ', num2str(bestCOerr(1,1))]);

else if bestCOerr(2,1) ~= bestCOR2(2,1)

        disp(['Best CO networks: ',['CO',num2str(bestCOerr(2,1))],[' and CO',num2str↙
(bestCOR2(2,1))]]);

        disp(['R2 for stability: ', ['CO',num2str(bestCOR2(2,1))] , [' --> ', num2str↙
(bestCOR2(1,1))]]);
        disp(['R2 for stability: ', ['CO',num2str(bestCOerr(2,1))] , [' --> ',num2str↙
(R2CO(1,bestCOerr(2,1)))]]);
```

```
        disp(['Mean error for stability: ',['CO',num2str(bestCOerr(2,1))] , [' --> ',↙
num2str(bestCOerr(1,1))]]);
        disp(['Mean error for stability: ',['CO',num2str(bestCOR2(2,1))] , [' --> ',↙
num2str(Error_CO(1,bestCOR2(2,1)))]]);
    end
end


%% Writing the results in an excel which has a scatter plot.

clear heading;

heading(1,1:3)={ 'Red n°' 'R2' 'Error Estabilidad'};
HC_writing=[transpose(R2HC(2,:)) , transpose(R2HC(1,:)) , transpose(Error_HC(1,:))];
HC_writing_2=[transpose(R2HC_pre(2,:)) , transpose(R2HC_pre(1,:)) , transpose↙
(Error_HC_pre(1,:))];
SOOT_writing=[transpose(R2SOOT(2,:)) , transpose(R2SOOT(1,:)) , transpose(Error_SOOT↙
(1,:))];
SOOT_writing_2=[transpose(R2SOOT_pre(2,:)) , transpose(R2SOOT_pre(1,:)) , transpose↙
(Error_SOOT_pre(1,:))];
CO_writing=[transpose(R2CO(2,:)) , transpose(R2CO(1,:)) , transpose(Error_CO(1,:))];
CO_writing_2=[transpose(R2CO_pre(2,:)) , transpose(R2CO_pre(1,:)) , transpose↙
(Error_CO_pre(1,:))];
xlswrite('DecisionError.xlsx',heading, 'HC');
xlswrite('DecisionError.xlsx',heading, 'HC_pre');
xlswrite('DecisionError.xlsx',HC_writing, 'HC', 'A2');
xlswrite('DecisionError.xlsx',HC_writing_2, 'HC_pre', 'A2');
xlswrite('DecisionError.xlsx',heading, 'SOOT');
xlswrite('DecisionError.xlsx',heading, 'SOOT_pre');
xlswrite('DecisionError.xlsx',SOOT_writing, 'SOOT', 'A2');
xlswrite('DecisionError.xlsx',SOOT_writing_2, 'SOOT_pre', 'A2');
xlswrite('DecisionError.xlsx',heading, 'CO');
xlswrite('DecisionError.xlsx',heading, 'CO_pre');
xlswrite('DecisionError.xlsx',CO_writing, 'CO', 'A2');
xlswrite('DecisionError.xlsx',CO_writing_2, 'CO_pre', 'A2');



%% Now the error of the predicted values from each network is tested: (predicted vs↙
real)

%_______ Testing error HC ________%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear yHc; clear newy; clear epsilon; clear realerror;
yHC=HC_pre95(TESTHC); %%% write in here the HC number of the best case
newy=exp(-yHC);

clear testy;
testy=HC95(TRAIN);  %%% write in here the HC number of the best case
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Differentiating steady, transient and pos-injection strategy (Plots)

posFrPulse4001 = find(FrPulse4==0); %returns the position of zero values
testy_pos_inj = testy(1:length(ALLDATA));
testy_pos_inj(posFrPulse4001)=[];        %selecting the values with pos injection↙
strategy
```

```
TRAINOUTHC_pos_inj = TRAINOUTHC(1:length(ALLDATA));
TRAINOUTHC_pos_inj(posFrPulse4001)=[];
testy_steady = testy(1:length(ALLDATA)); %selecting the values of steady conditions
TRAINOUTHC_steady = TRAINOUTHC(1:length(ALLDATA));

rangeHC=size(TESTOUTHC);
for j=1:rangeHC(1)
       epsilon(j)=TESTOUTHC(j)/yHC(j); %not used%
       realerror(j)=(newy(j)-exp(-TESTOUTHC(j)))/exp(-TESTOUTHC(j));
end

% Determination Coefficient
HC = fitlm(exp(-TRAINOUTHC),exp(-testy));
r_srq_hc = HC.Rsquared.Ordinary;


figure('Name','HC','NumberTitle','off');
subplot(2,3,3)
boxplot(abs(realerror)); ylabel('Relative error - testing set'); xlabel([num2str↙
(rangeHC(1)) ' test samples']); title('HC');

subplot(2,3,[1,2,4,5])
plot(exp(-TRAINOUTHC),exp(-testy),'b.'); hold on;  %testy are the predicted values%%
plot(exp(-TRAINOUTHC_steady),exp(-testy_steady),'r.'); hold on;
plot(exp(-TRAINOUTHC_pos_inj),exp(-testy_pos_inj),'g.'); hold on;
plot(exp(-TRAINOUTHC), exp(-TRAINOUTHC),'black'); xlabel('Real emission'); ylabel↙
('Predicted emission'); title('HC'); % TRAINOUTHC are the measured values↙
(experimental data)%
legend('Transient','Steady', 'Steady (post-inj)', 'Location','northwest')
annotation('textbox',[0.7 0.02 0.1 0.2],'String',{'Determination Coef.',↙
r_srq_hc},'FitBoxToText','on');
annotation('textbox',[0.7 0.02 0.1 0.4],'String',{'MSE Validation Set',↙
error_hc},'FitBoxToText','on');
savefig('HC.fig')


%%

%_______ Testing error SOOT ________%

clear epsilon; clear realerror; clear newy; clear ySOOT

rangeSOOT=size(TESTOUTSOOT);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ySOOT=SOOT11(TESTSOOT); %%% write in here the SOOT number of the best case
newy=exp(-ySOOT);
for j=1:rangeSOOT(1)

       epsilon(j)=TESTOUTSOOT(j)/ySOOT(j);
       realerror(j)=(newy(j)-exp(-TESTOUTSOOT(j)))/exp(-TESTOUTSOOT(j));
end

clear testy; clear testy_pos_inj; clear testy_pos_inj;
testy=SOOT11(TRAINSOOT);%%% write in here the SOOT number of the best case

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Differentiating steady, transient and pos-injection strategy (Plots)
```

```
FrPulse4_soot = FrPulse4;
FrPulse4_soot(posYSOOT01)=[]; %exclude the values related to zero values of Soot
posFrPulse4001 = find(FrPulse4_soot==0);
testy_pos_inj = testy(1:length(ALLDATA2));
testy_pos_inj(posFrPulse4001)=[];        %selecting the values with pos injection
strategy
TRAINOUTSOOT_pos_inj = TRAINOUTSOOT(1:length(ALLDATA2));
TRAINOUTSOOT_pos_inj(posFrPulse4001)=[];
testy_steady = testy(1:length(ALLDATA2)); %selecting the values of steady conditions
TRAINOUTSOOT_steady = TRAINOUTSOOT(1:length(ALLDATA2));

% Estimating the Coefficient of Determination

SOOT = fitlm(exp(-TRAINOUTSOOT),exp(-testy));
r_srq_soot = SOOT.Rsquared.Ordinary;

figure('Name','Soot','NumberTitle','off');
subplot(2,3,3)
boxplot(abs(realerror)); ylabel('Relative error - testing set'); xlabel([num2str
(rangeSOOT(1)) ' test samples']); title('SOOT');

subplot(2,3,[1,2,4,5])
plot(exp(-TRAINOUTSOOT),exp(-testy),'b.'); hold on;
plot(exp(-TRAINOUTSOOT_steady),exp(-testy_steady),'r.'); hold on;
plot(exp(-TRAINOUTSOOT_pos_inj),exp(-testy_pos_inj),'g.'); hold on;
plot(exp(-TRAINOUTSOOT), exp(-TRAINOUTSOOT), 'black'); xlabel('Real emission'); ylabel
('Predicted emission'); title('Soot');
legend('Transient','Steady', 'Steady (post-inj)', 'Location','northwest')
annotation('textbox',[0.7 0.02 0.1 0.2],'String',{'Determination Coef.',
r_srq_soot},'FitBoxToText','on');
annotation('textbox',[0.7 0.02 0.1 0.4],'String',{'MSE Validation Set',
error_soot},'FitBoxToText','on');
savefig('Soot.fig')



%_______ Testing error CO ________%

clear epsilon; clear realerror; clear yCO; clear newy


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
yCO=CO22(TESTCO); %%% write in here the CO number of the best case
newy=exp(-yCO);
rangeCO=size(TESTOUTCO);

clear testy; clear testy_pos_inj; clear posFrPulse4001
testy=CO22(TRAIN); %%% write in here the CO number of the best case
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


posFrPulse4001 = find(FrPulse4==0); %returns the position of zero values
testy_pos_inj = testy(1:length(ALLDATA));
testy_pos_inj(posFrPulse4001)=[];        %selecting the values with pos injection
strategy
TRAINOUTCO_pos_inj = TRAINOUTCO(1:length(ALLDATA));
TRAINOUTCO_pos_inj(posFrPulse4001)=[];
testy_steady = testy(1:length(ALLDATA)); %selecting the values of steady conditions
TRAINOUTCO_steady = TRAINOUTCO(1:length(ALLDATA));
```

```
CO = fitlm(exp(-TRAINOUTCO),exp(-testy));
r_srq_co = CO.Rsquared.Ordinary;

for j=1:rangeCO(1)

       epsilon(j)=TESTOUTCO(j)/yCO(j);
       realerror(j)=(newy(j)-exp(-TESTOUTCO(j)))/exp(-TESTOUTCO(j));
end

figure('Name','CO','NumberTitle','off');
subplot(2,3,3)
boxplot(abs(realerror)); ylabel('Relative error - testing set'); xlabel([num2str
(rangeCO(1)) ' test samples']); title('CO');

subplot(2,3,[1,2,4,5])
plot(exp(-TRAINOUTCO),exp(-testy),'b.'); hold on;
plot(exp(-TRAINOUTCO_steady),exp(-testy_steady),'r.'); hold on;
plot(exp(-TRAINOUTCO_pos_inj),exp(-testy_pos_inj),'g.'); hold on;
plot(exp(-TRAINOUTCO), exp(-TRAINOUTCO),'black'); xlabel('Real emission'); ylabel
('Predicted emission'); title('CO');
legend('Transient','Steady', 'Steady (post-inj)', 'Location','northwest')
annotation('textbox',[0.7 0.02 0.1 0.2],'String',{'Determination Coef.',
r_srq_co},'FitBoxToText','on');
annotation('textbox',[0.7 0.02 0.1 0.4],'String',{'MSE Validation Set',
error_co},'FitBoxToText','on');

savefig('CO.fig')


%% Predicting: emissions for the reguar data %%

%%%%% The number of the best case for each network must be put for each
%%%%% emission, for instance if the best cases are HC2, SOOT23 and CO$,
%%%%% those three shall be put in both this and the next section
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
yHC2=HC76(TRAIN);%number here        % sub 2 only to differ different data
yHC=exp(-yHC2);                      %yHC are the predicted values based only on
inputs
yHC_exp=exp(-TRAINOUTHC);

yCO2=CO40(TRAIN);%number here
yCO=exp(-yCO2);
yCO_exp=exp(-TRAINOUTCO);

ySOOT2=SOOT18(TRAINSOOT);%number here
ySOOT=exp(-ySOOT2);
ySOOT_exp=exp(-TRAINOUTSOOT);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Predicting: emissions for perturbed data %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
yHCvar2=HC76(TRAINv);%number here           % sub 2 only to differ differents datas
yHCvar=exp(-yHCvar2);                       % sub var is related to perturbed data

yCOvar2=CO40(TRAINv);%number here
yCOvar=exp(-yCOvar2);

ySOOTvar2=SOOT18(TRAINv2);%number here
```

```
ySOOTvar=exp(-ySOOTvar2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% Differentiating steady, transient and pos-injection strategy (Plots)

posFrPulse4001 = find(FrPulse4==0); %returns the position of zero values

%HC
yHCvar2_pos_inj = yHCvar2(1:length(ALLDATAv));
yHCvar2_pos_inj(posFrPulse4001)=[];        %selecting the values with pos injection↙
strategy
yHC2_pos_inj = yHC2(1:length(ALLDATA));
yHC2_pos_inj(posFrPulse4001)=[];
yHCvar2_steady = yHCvar2(1:length(ALLDATAv)); %selecting the values of steady↙
conditions
yHC2_steady = yHC2(1:length(ALLDATA));

%CO
yCOvar2_pos_inj = yCOvar2(1:length(ALLDATAv));
yCOvar2_pos_inj(posFrPulse4001)=[];        %selecting the values with pos injection↙
strategy
yCO2_pos_inj = yCO2(1:length(ALLDATA));
yCO2_pos_inj(posFrPulse4001)=[];
yCOvar2_steady = yCOvar2(1:length(ALLDATAv)); %selecting the values of steady↙
conditions
yCO2_steady = yCO2(1:length(ALLDATA));

%Soot
clear posFrPulse4001;

FrPulse4_soot = FrPulse4;
FrPulse4_soot(posYSOOT01)=[]; %exclude the values related to zero values of Soot
posFrPulse4001 = find(FrPulse4_soot==0);
ySOOTvar2_pos_inj = ySOOTvar2(1:length(ALLDATA2v));
ySOOTvar2_pos_inj(posFrPulse4001)=[];        %selecting the values with pos injection↙
strategy
ySOOT2_pos_inj = ySOOT2(1:length(ALLDATA2));
ySOOT2_pos_inj(posFrPulse4001)=[];
ySOOTvar2_steady = ySOOTvar2(1:length(ALLDATA2v)); %selecting the values of steady↙
conditions
ySOOT2_steady = ySOOT2(1:length(ALLDATA2));


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%introduce here the number of the function that's been used:%
bH=76;%--- Best HC
bS=3;%--- Best SOOT
bC=39;%--- Best CO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plotting the results

figure('Name','Predicted Results','NumberTitle','off');
subplot(2,2,1)
plot(yHC,yHCvar,'b.'); hold on;
plot(exp(-yHC2_steady),exp(-yHCvar2_steady),'r.'); hold on;
plot(exp(-yHC2_pos_inj),exp(-yHCvar2_pos_inj),'g.'); hold on;
plot(yHC,yHC); %xlabel('Original predicted emission'); ylabel('Perturbed emission');
title('Predicted x Perturbed - HC');
```

```
legend('Transient','Steady', 'Steady (post-inj)', 'Location','northwest')

subplot(2,2,2)
plot(yCO,yCOvar,'b.'); hold on;
plot(exp(-yCO2_steady),exp(-yCOvar2_steady),'r.'); hold on;
plot(exp(-yCO2_pos_inj),exp(-yCOvar2_pos_inj),'g.'); hold on;
plot(yCO,yCO);
title('Predicted x Perturbed - CO');
legend('Transient','Steady', 'Steady (post-inj)', 'Location','northwest')

subplot(2,2,3)
plot(ySOOT,ySOOTvar,'b.'); hold on;
plot(exp(-ySOOT2_steady),exp(-ySOOTvar2_steady),'r.'); hold on;
plot(exp(-ySOOT2_pos_inj),exp(-ySOOTvar2_pos_inj),'g.'); hold on;
plot(ySOOT,ySOOT);
title('Predicted x Perturbed - SOOT');
legend('Transient','Steady', 'Steady (post-inj)', 'Location','northwest')
str = {'Variation: 1.03*YO2TDC', '  ', 'Coefficient of Determination:','HC',R2HC(1,↙
bH), '  ','CO',R2CO(1,bC),  '  ','Soot',R2SOOT(1,bS)};
annotation('textbox',[0.6 0.3 0.1 0.2],'String',str,'FitBoxToText','on');

savefig('Pred_all.fig')


%% XML

%%%%%%%%%%%%%%% BEFORE RUNNING THIS SECTION %%%%%%%%%%%%%%%%
% If bH, bS and bC havent been imputed, imput them now  %

%introduce here the number of the function that's been used:%
bH=95;%--- Best HC
bS=3;%--- Best SOOT
bC=39;%--- Best CO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %Writing the XML file containing all the useful data

    %Open the xml file

w=fopen('AnnV2.0.xml','wt');

firstline=['           <NeuralNetworks Version="' Version '">\n'];

fprintf(w,firstline);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% CO network

    fprintf(w,'            <NetworkCO nbLayers="%i"',size(networkCO,2));%1
    fprintf(w,' nbInputs="%i">\n', size(minOUTCO,1));

        % Ranges

        fprintf(w,'             <Ranges>\n');%2
        for i=1:size(minOUTCO,1)
            fprintf(w,'              <Range_input min="%16.10e"',minOUTCO(i,1));%3↙
%%%%%
            fprintf(w,' max="%16.10e" />\n',maxOUTCO(i,1));%%%%%
        end
        fprintf(w,'              <Range_output min="%16.10e"',minmaxCO(1,1));%3
        fprintf(w,' max="%16.10e" />\n', minmaxCO(2,1));
```

```
        fprintf(w,'            </Ranges>\n');%2

        % Layers

        fprintf(w,'            <Layers>\n');%2
            for i=1:size(networkCO,2)+1
                 fprintf(w,'              <Layer ');%3
                 fprintf(w,'nbNodes="%i">\n',nbNodesCO(i));
                    for j=1:size(WCO{i},2)
                        fprintf(w,'               <Weight value="%16.10e" />\n',WCO{i,↙
bC}(j));%4    %Comment what bc is being used for.
                    end
                    for j=1:size(bCO{i},2)
                        fprintf(w,'               <Bias value="%16.10e" />\n',bCO{i,↙
bC}(j));%4
                    end

                 fprintf(w,'              </Layer>\n');%3
            end
        fprintf(w,'            </Layers>\n');%2

    fprintf(w,'           </NetworkCO>\n');%1


    %%%%%%%%%%%%%%%%%%%%%%%%%%%% HC Network

    fprintf(w,'           <NetworkHC nbLayers="%i"',size(networkHC,2));%1
    fprintf(w,' nbInputs="%i">\n', size(minOUTHC,1));

        % ranges

        fprintf(w,'            <Ranges>\n'); %2
        for i=1:size(minOUTHC,1)
            fprintf(w,'               <Range_input min="%16.10e"',minOUTHC(i,1));%3
            fprintf(w,' max="%16.10e" />\n',maxOUTHC(i,1));
        end
        fprintf(w,'             <Range_output min="%16.10e"',minmaxHC(1,1));%3
        fprintf(w,' max="%16.10e" />\n', minmaxHC(2,1));
        fprintf(w,'            </Ranges>\n');

        %Layers

        fprintf(w,'            <Layers>\n');%2
            for i=1:size(networkHC,2)+1
                 fprintf(w,'              <Layer ');%3
                 fprintf(w,'nbNodes="%i">\n',nbNodesHC(i));

                    for j=1:size(WHC{i},2)
                        fprintf(w,'               <Weight value="%16.10e" />\n',WHC{i,↙
bH}(j));%4
                    end


                    for j=1:size(bHC{i},2)
                        fprintf(w,'               <Bias value="%16.10e" />\n',bHC{i,↙
bH}(j));
                    end

                 fprintf(w,'              </Layer>\n');%3
            end
```

```
        fprintf(w,'             </Layers>\n');%2

    fprintf(w,'            </NetworkHC>\n');%1



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SOOT Network

    fprintf(w,'            <NetworkSOOT nbLayers="%i"',size(networkSOOT,2));%1
    fprintf(w,' nbInputs="%i">\n', size(minOUTSOOT,1));

        %ranges

        fprintf(w,'             <Ranges>\n'); %2
        for i=1:size(minOUTSOOT,1)
            fprintf(w,'              <Range_input min="%16.10e"',minOUTSOOT(i,1));%3
            fprintf(w,' max="%16.10e" />\n',maxOUTSOOT(i,1));
        end
        fprintf(w,'              <Range_output min="%16.10e"',minmaxSOOT(1,1));%3
        fprintf(w,' max="%16.10e" />\n', minmaxSOOT(2,1));
        fprintf(w,'             </Ranges>\n');

        %Layers

        fprintf(w,'             <Layers>\n');%2
            for i=1:size(networkSOOT,2)+1
                 fprintf(w,'              <Layer ');%3
                 fprintf(w,'nbNodes="%i">\n',nbNodesSOOT(i));

                    for j=1:size(WSOOT{i},2)
                        fprintf(w,'               <Weight value="%16.10e" />\n',WSOOT↙
{i,bS}(j));%4
                    end


                    for j=1:size(bSOOT{i},2)
                        fprintf(w,'               <Bias value="%16.10e" />\n',bSOOT{i,↙
bS}(j));
                    end

                 fprintf(w,'             </Layer>\n');%3
            end
        fprintf(w,'             </Layers>\n');%2

    fprintf(w,'            </NetworkSOOT>\n');%1

fprintf(w,'          </NeuralNetworks>\n');

fclose(w);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%↙
%%%%%%%%
```