

Capítulo 1

Introducción

El motor de inducción es un motor de corriente alterna que es usado ampliamente en la industria debido a su bajo coste y a su alto rendimiento y fiabilidad. La función principal de los motores de inducción es generar energía mecánica a partir de energía eléctrica y se usan principalmente en electrodomésticos, aparatos de uso cotidiano y para el desplazamiento en vehículos eléctricos. Sin embargo, como el modelo dinámico del motor de inducción es no lineal, el control del motor de inducción es un problema complejo y ha obtenido mucha atención. El desarrollo de métodos de control para modelos no lineales ha crecido en las últimas tres décadas y se puede agrupar en: métodos de linealización con retroalimentación, funciones de control Lyapunov y métodos pasivos. Sin embargo, el que más ha destacado ha sido el control de campo orientado [5, 11]. La finalidad del control de campo orientado consiste en intentar conseguir una especie de control lineal haciendo independientes la corriente que produce la variación del flujo magnético con la corriente que produce el par del motor. Para lograr esto, se crea una referencia de circuitos ficticias con dos bobinas en cuadratura, formando un ángulo de 90^0 , en el estator y equivalente a las tres bobinas reales. Es decir, se transforma el sistema trifásico de corrientes a un sistema bifásico de corrientes rotatorio que gira sincrónicamente con el campo magnético del rotor.

Existen métodos para diseñar un controlador para un motor de inducción de manera rápida cuando se conoce perfectamente los parámetros del motor y cuando hay cierta incertidumbre asociada a la estimación de los parámetros. Los modelos basados en métodos de control para motores de inducción solo requieren tener un conocimiento apropiado sobre cuatro parámetros eléctricos independientes del motor. Estos cuatro parámetros son diferentes según el marco de referencia que hayamos escogido para representar el motor de inducción. Si optamos por un marco de referencia bifásico estacionario (ejes $\alpha\beta$), los cuatro parámetros son: $i_{\alpha s}$, $i_{\beta s}$, $\lambda_{\alpha r}$ y $\lambda_{\beta r}$ que se corresponden

con la intensidad de la corriente en las bobinas del estator medidas en los ejes α y β , respectivamente; y el flujo magnético normal al plano de las bobinas del rotor medido en los ejes α y β , respectivamente. Por otra parte, si elegimos un marco de referencia bifásico rotatorio (ejes dq), es decir que gira solidariamente con el campo magnético rotatorio generado en el rotor, los cuatro parámetros serían: λ_{qs} , λ_{ds} , λ_{qr} y λ_{dr} que se corresponderían con el flujo magnético que atraviesa las bobinas del estator y del rotor medido en los ejes dq .

Tradicionalmente se han utilizado controladores que contienen 3 bloques: *proporcional, integral y derivativo* (PID),

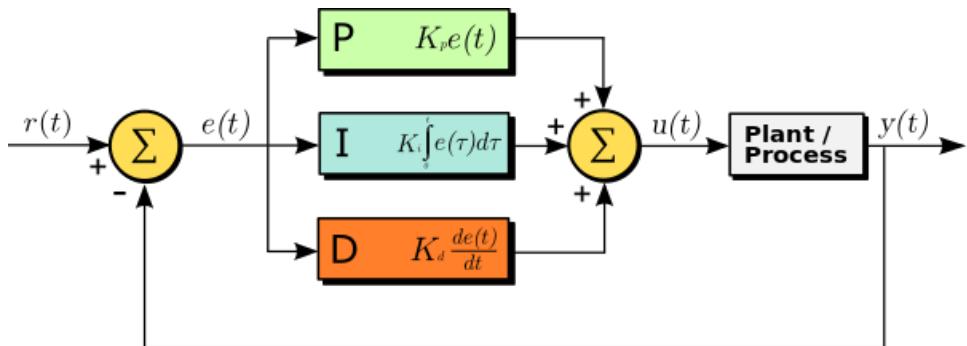


Figura 1.1: Sistema controlado por un PID. Imagen obtenida de https://en.wikipedia.org/wiki/PID_controller

para conseguir operar rápidamente sobre los 4 cuadrantes, un arranque suave y una aceleración del motor de inducción. Estos controladores PID poseen una gran cantidad de ventajas tales como su fácil diseño, su estructura de control simple y su bajo coste. No obstante, también poseen importantes inconvenientes, tales como exhibir una pobre respuesta respecto a la variación de parámetros del sistema y contra perturbaciones de la carga cuando operan a baja velocidad y la no reusabilidad ya que el rendimiento del PID se suele ver muy afectado por las diferencias entre el proceso real y el modelo.

En los últimos años ha habido un interés creciente en el uso de métodos de aplicaciones inteligentes para manejar la no linealidad e incertidumbre del motor de inducción, entre los que destaca los algoritmos difusos, los algoritmos genéticos y las redes neuronales. Se han propuesto métodos de control basados en redes neuronales para aprender modelos que puedan superar los problemas asociados a los modelos tradicionales. Se han usado diferentes métodos como el uso de redes neuronales radiales para aprender y conocer la relación existente entre las corrientes y los flujos o clasificadores basados en redes neuronales para el diagnóstico de errores en motores eléctricos. Una de las mejores propuestas para el motor de inducción ha sido usar un

controlador que combine las ventajas de los controladores PID y las redes neuronales [22]. En sistemas de control, las redes neuronales convencionales son una herramienta para el diseño de controladores [9], identificación de sistemas[4] y el auto-ajuste[14].

Uno de los mayores problemas al usar redes neuronales artificiales es el cálculo de los hiperparámetros ya que se sigue un método de optimización como gradiente descendente. Por esto, se propone complementar el alto coste computacional de entrenamiento de las redes neuronales con actualizaciones de los hiperparámetros recogidos manualmente con la ayuda de herramientas eficientes para la simulación de modelos ciberfísicos. El algoritmo de optimización que se usa en el entrenamiento de redes neuronales en la industria es el mencionado gradiente descendente debido a su sencillez y facilidad de implementación. Este algoritmo consiste en el ajuste de los distintos pesos de la red neuronal para llegar a una convergencia del error y además permite la retroalimentación o propagación hacia atrás del error que comete la red neuronal, lo que hace posible que todas las neuronas conozcan su contribución al error en función de sus pesos y los vayan ajustando. Las evaluaciones de la función objetivo da como resultado un vector de hipergradiente, es decir, uno que aprende de varios parámetros optimizados para combinar diferentes niveles de adaptación en vez de considerar valores individuales como se produce en la mayoría de los métodos.

Hoy en día podemos obtener el gradiente de selección del modelo en un espacio de tiempo prudencial gracias a las tarjetas gráficas y a las computadoras modernas con un alto nivel de paralelismo en los cálculos, algo que es crucial para el ajuste de los hiperparámetros de los modelos ciberfísicos. Gracias a la computación paralela, actualmente podemos manejar una gran cantidad de hiperparámetros de un modelo implementando métodos basados en gradiente descendente de manera eficiente y más específicamente usando redes neuronales [13, 19]. Es posible mejorar el proceso completo de entrenamiento de un modelo ciberfísico optimizado con una red neuronal a través de una actualización offline y adaptativa de los hiperparámetros. Ya sea midiendo las diferentes componentes ciberfísicas del modelo a estudiar o realizando una validación manual desde un programa que trate de simular el modelo ciberfísico. **En este trabajo se considerará la opción de la simulación del motor de inducción**, utilizando un modelo de motor simulado ya probado [8]. Aunque este método se podría considerar muy específico y de aplicación a un modelo concreto de sistema ciberfísico, sin embargo permite ajustar una gran cantidad de hiperparámetros del modelo lo que conlleva una gran mejora respecto a otros métodos explorados hasta la fecha.

El sobreajuste es una condición que ocurre cuando un modelo clasifica o predice con un alto grado de acierto con los datos del conjunto de entre-

namiento pero no consiguen una tasa de acierto tan alta para los nuevos datos. Es decir, provoca que el modelo «memorice» los datos de entrenamiento y no clasifique o prediga correctamente los nuevos datos. Cuando se utilizan redes neuronales junto a un conjunto de validación finito, el cálculo de los hiperparámetros podría sufrir este sobreajuste. Una posibilidad para disminuir el sobreajuste es usar validación cruzada, una técnica que nos permite escoger de manera aleatoria los datos disponibles para cada función que se vaya a evaluar. Se ha demostrado que la validación cruzada mejora la precisión y la exhaustividad de los modelos ciberfísicos.

Para resolver estos problemas tan complejos y otros problemas con una gran cantidad de datos ha surgido el aprendizaje profundo y su implementación a partir de las redes neuronales convencionales. Esto ha permitido crear redes neuronales profundas, es decir, redes neuronales con un número mayor de capas no lineales y neuronas que consiguen utilizar más parámetros, ya que se modera el sobreajuste.

Si lo comparamos con las redes neuronales convencionales, el factor diferencial del aprendizaje profundo se encuentra en un mayor número de capas ocultas y neuronas que consiguen una mejora en el rendimiento del aprendizaje. Gracias a sus características, los problemas complejos y de gran cantidad de datos, que las redes neuronales convencionales no podían resolver, sí pueden resolverse a través de algoritmos de aprendizaje profundo. Sin embargo, las redes neuronales profundas no son muy utilizadas todavía pero son más efectivas en la resolución de los mencionados problemas.

En sistemas híbridos [8], el comportamiento continuo descrito por un sistema de ecuaciones diferenciales debe cambiar también tras acontecimientos de eventos discretos. Este enfoque permite realizar especificaciones muy compactas y flexibles para sistemas híbridos complejos. Sin embargo, hay muy pocas herramientas que soporten esta clase de enfoque de diseño de sistemas híbridos a día de hoy.

Para la implementación de sistemas ciberfísicos se recurre a herramientas como MATLAB/Simulink \circledR para la creación y simulación del modelo junto a Stateflow \circledR que proporciona un lenguaje gráfico que incluye diagramas de transición de estado y diagramas de flujo que nos permite saltar de estado pudiendo representar un comportamiento híbrido, que son ampliamente utilizadas en la industria actual para complementar con éxito los métodos de aprendizaje automático. Actualmente, se considera que el uso de herramientas, como las mencionadas, que pueden utilizarse para acelerar la determinación de los hiperparámetros de una red neuronal, aplicada al modelado del control de sistemas ciberfísicos, son una base firme para los métodos de aprendizaje automático en el entorno industrial y pueden ser considerados como una alternativa menos compleja que la utilización de *metaheurísticas* para el desarrollo de nuevos algoritmos [8] de control.

1.1. Objetivos previstos

Los objetivos del trabajo son el diseño correcto de un sistema dinámico no lineal, como un motor de inducción controlado por un PID en bucle cerrado, utilizando Simulink/Matlab[©] con modelos de Machine Learning (DeepLearning Toolbox[©]), la aplicación práctica de los métodos de aprendizaje automático para sustituir el mencionado de control industrial que hasta ahora se conseguía generalmente implementando un PID, y que en este trabajo proponemos sustituir utilizando librerías de redes neuronales de un tipo concreto feed-forward, recurrentes, NARX, etc. que facilita el DeepLearning Toolbox de Simulink, y obtener series temporales de valores de variables que representan magnitudes físicas de referencia (corrientes, tensiones, flujos, etc.) del sistema ciberfísico que representan los aspectos funcionales y dinámicos del sistema físico elegido para el realizar el estudio, en este caso, un motor de inducción.

1.2. Justification

El motor de inducción es el motor eléctrico de corriente alterna más usado a nivel industrial por su fiabilidad y su bajo coste. Sin embargo, debido a que su modelo dinámico es no lineal, el control de un motor de inducción es un problema algo complejo. La técnica más usada para controlar el motor de inducción ha sido el uso de controladores PID ya que poseen una estructura de control simple, un fácil diseño y un coste económico bajo. No obstante, muestran resultados muy pobres cuando se produce perturbaciones en el par de carga o variaciones en los parámetros cuando la velocidad es baja y no son reutilizables, solo sirven para un modelo concreto.

Por estas razones, en este trabajo, primeramente se usarán redes neuronales artificiales para estimar dos parámetros del motor de inducción y tras ello, se sustituirá un controlador PID por una red neuronal. Si obtenemos los resultados esperados, podremos sustituir los controladores PID que son ampliamente usados en los motores de inducción por redes neuronales, lo que permitirá crear nuevas estructuras de control con las mismas ventajas que los controladores PID pero sin sus desventajas. A nivel industrial repercutirá, por ejemplo, en cualquier dispositivo y máquina que hagan uso de motores de inducción, por ejemplo, en la conducción automática de vehículos eléctricos autónomos, para poder organizar de mejor manera el tráfico.

Capítulo 2

Antecedentes

Existen numerosos trabajos de investigación que han sido enfocados a la resolución de los problemas relacionados con el control de motores de inducción utilizando para ello redes neuronales, algunos de estos son los que se resumen a continuación.

2.1. Modelado de los sistemas de referencia a utilizar para resolver el problema del motor de inducción

En [20], se cuenta la efectividad de la teoría de los marcos de referencia para analizar el rendimiento de un motor de inducción debido a que su estudio y modelado con reguladores electromagnéticos presenta gran dificultad al estar en un marco trifásico. Se presenta, paso por paso, cómo implementar un motor de inducción usando transformaciones sobre el eje dq_0 en las variables del estator y del rotor que se encuentran en un marco de referencia arbitrario. A partir de las ecuaciones de un motor de inducción, se generaliza un motor de inducción de tres fases que se desarrolla e implementa de forma sencilla. Los resultados proveen evidencias que refutan la teoría de los marcos de referencia.

2.2. Distintos enfoques para implementar el controlador del motor

En [18], se propone una técnica para controlar la velocidad de un motor de inducción: el control predictivo del modelo (MPC). El diseño del controlador

MPC tiene en cuenta el efecto de la incertidumbre debido a que la variación de los parámetros del motor, tales como las perturbaciones de la carga pueden disminuir su rendimiento. Se compara el rendimiento del sistema junto al del controlador MPC con el mismo sistema usando un controlador PID tradicional. Se obtiene una mejor respuesta y una gran robustez frente a la incertidumbre de los parámetros y de las perturbaciones de la carga pero los controladores MPC son más complejos y más dependientes del modelo que los controladores PID [7].

En [17], se propone un método de control escalar junto con la técnica de Control de Frecuencia Variable (VFC) para controlar la velocidad del motor de inducción. El método consiste en variar la frecuencia del motor de inducción, que opera con diferentes pares de cargas, para que alcance la velocidad de referencia. Se utilizan métodos de curvas de ajuste junto a soluciones conseguidas con métodos numéricos para proporcionar la variación de la frecuencia requerida. Se usan métodos polinomiales, Gaussianos y de Fourier. Los resultados obtenidos muestran que la respuesta es rápida a diferentes velocidades y par de cargas y provee un control de la velocidad estable y preciso.

En [1], se propone otro método para controlar la velocidad del motor de inducción: un sistema de inferencia adaptativo que combina las redes neuronales con la lógica difusa (Neuro-Fuzzy System). Este sistema híbrido combina la capacidad adaptativa de las redes neuronales junto con la aproximación cualitativa de la lógica difusa. Se comparan los resultados de un controlador PID convencional y del controlador propuesto. Los resultados muestran que el controlador propuesto posee características adaptativas que hacen que el controlador más robusto para un rango diverso de condiciones de carga y mejora el tiempo de respuesta de la planta.

2.3. Aplicación de redes neuronales a la resolución de este problema

En [8], se presenta el actual interés creciente de los métodos de aprendizaje automático junto con sus principales problemas en la computación de los hiperparámetros de las redes neuronales y el impacto de los hiperparámetros en el rendimiento de un modelo ciber-físico (motor de inducción) con cuatro estimadores basados en redes neuronales FeedForward y NARX utilizando gradiente descendente. El objetivo es demostrar que los valores optimizados para los hiperparámetros pueden validarse a través de simular el modelo en MATLAB/Simulink. Para el entrenamiento de las redes neuronales se usa

el voltaje y las corrientes y se quiere estimar la velocidad del rotor y el par electromagnético. Los resultados demuestran la efectividad de los estimadores de redes neuronales obtenidos y pueden convertirse en bloques para usarse en un modelo ciber-físico diseñado en Simulink.

En [2], se presenta el uso de redes neuronales de tipo FeedForward para una estimación precisa de la velocidad y de los flujos en un motor de inducción para aplicaciones industriales. Se diseñan dos redes neuronales, una para la velocidad del rotor y otra para los flujos del rotor y del estator. Los datos de entrenamiento de entrada son los voltajes y las corrientes mientras que los datos de salida son la velocidad y los flujos medidos. Las redes neuronales superan el problema de variación de los parámetros y prueba su efectividad, su desarrollo se ha hecho a través de MATLAB/Simulink.

2.4. Aplicación del aprendizaje profundo a las redes neuronales para resolver el problema de control del motor de inducción

En [15], se presenta un nuevo modelo para reemplazar el controlador PID usado en los motores de inducción y que aplica el aprendizaje profundo aplicado a redes neuronales. Al aumentar el número de capas ocultas y de neuronas se mejora el rendimiento de las redes neuronales y pueden aplicarse a problemas más grandes y complejos que las redes neuronales convencionales. Se diseña un controlador a través de redes neuronales profundas a partir de las entradas y salidas de un controlador PID convencional, donde se utiliza el algoritmo ADBN para diseñar este controlador.

2.5. Mi trabajo

En este trabajo, debido al interés creciente en el desarrollo del «Internet de las cosas» y del extenso uso de los motores de inducción en la industria, se presenta **el diseño de un modelo del motor de inducción en el marco de referencia $\alpha\beta$** en MatLab/Simulink y la implementación de cuatro estimadores basados en redes neuronales (dos prealimentadas y dos recurrentes) para la velocidad del rotor y los flujos magnéticos del rotor a partir de los voltajes y de las corrientes para conseguir demostrar que el uso de redes neuronales prealimentadas y recurrentes es válido para la predicción de los hiperparámetros en un motor de inducción a pesar de las perturbaciones en el par de carga.

Con los resultados obtenidos, se concluye que es posible sustituir un

controlador PID por una red neuronal, tal como otros autores, anteriormente mencionados, han intentado llevar a cabo en otros trabajos relacionados. Los resultados que se han obtenido ratifican la efectividad de las redes neuronales como estimadoras y la sustitución del controlador PID por una red neuronal.

Capítulo 3

Métodos y materiales

3.1. Materiales

En este trabajo se ha usado el software de MATLAB (versión 9.13.0.2166757 (R2022b) Update 4) junto con Simulink y el paquete de herramientas DeepLearningToolbox. El modelo del motor de inducción simulado diseñado en Simulink, que se ha usado como caso de estudio, como el archivo que contiene el valor de los parámetros del motor de inducción junto con las instrucciones para la creación de los datos de entrada y salida de las redes y todas las gráficas e imágenes generadas se pueden ver y descargar en la siguiente dirección: <https://github.com/Gerardo2000/TFG-modelo-motor>.

3.2. El motor de inducción

Un motor de inducción es un motor eléctrico de corriente alterna que se caracteriza porque el rotor gira a una velocidad diferente a la del campo magnético generado por el estator. El motor de inducción está formado por un rotor ardilla o bobinado) y un estator donde se hallan las bobinas inducadoras. El motor funciona aplicando el principio físico de inducción mutua por el paso de una corriente eléctrica a través de un flujo magnético variante.

Para generar el campo magnético giratorio en el estator de un motor de inducción se necesitan 3 bobinas tal que cada una de ellas conduce una corriente con una fase de voltaje que difiere $\frac{2\pi}{3}$ tanto en tiempo como en espacio con la siguiente bobina.

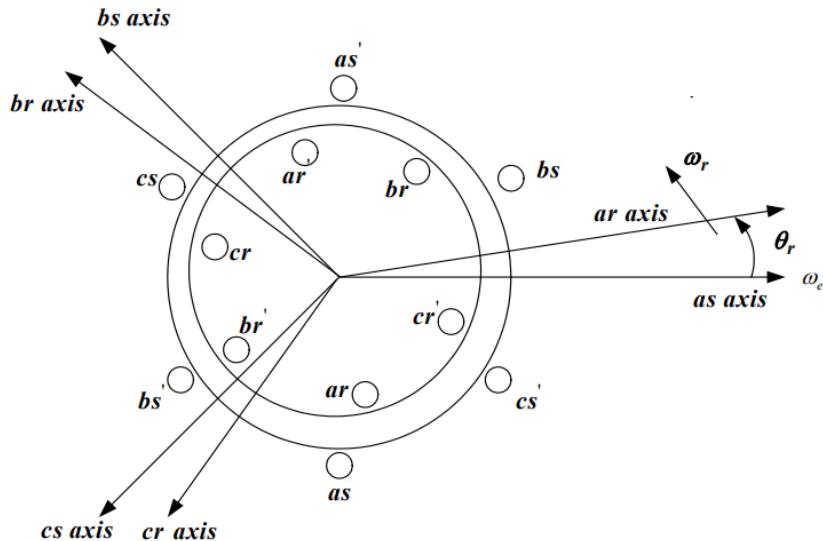


Figura 3.1: Motor de inducción trifásico simétrico con dos polos. Imagen obtenida de CHAPTER 3. MODEL OF A THREE-PHASE INDUCTION MOTOR.

De esta manera, se genera un campo magnético giratorio que envuelve al rotor y es el que va a inducir una tensión eléctrica en él según la ley de Faraday.

La Ley de Faraday o Ley de inducción electromagnética de Faraday establece que la tensión inducida en un circuito cerrado es directamente proporcional y de signo contrario (se opone a la corriente previa del circuito) a la rapidez con la que cambia en el tiempo el flujo magnético que atraviesa una superficie cualquiera con el circuito como borde.

$$\epsilon = -\frac{d\Phi_B}{dt} \quad (3.1)$$

Para formar los pares de polos magnéticos que producen el movimiento del motor, se necesita que el devanado del rotor conduzca una corriente para generar el flujo magnético. Al cortocircuitar el devanado, se induce una corriente y se produce la fuerza electromagnética. El motor girará a una velocidad (ω_r) debido a la fuerza electromagnética generada por los cambios de dirección del campo magnético B que rota a una velocidad angular sincrónica (ω_e) provocada por el devanador del estator.

Los ejes magnéticos de las bobinas del rotor tienen un desfase de ángulo θ_r ($\theta_r = \omega_r \cdot t$) respecto a los ejes magnéticos de las bobinas del estator.

Por la ley de Faraday sabemos que cuando el flujo magnético varía con el

tiempo, se produce la fuerza electromagnética lo que se puede traducir como un momento capaz de mover una carga. Sin embargo, cuando $\omega_r = \omega_e$, es decir, si la velocidad del giro del rotor coincide con la velocidad rotatoria del campo magnético del estator no se inducirá ninguna fuerza electromagnética ni se creará ningún momento. Por eso es necesario que exista una pequeña diferencia entre ω_r y ω_e para que se genere una fuerza electromagnética y se origine un momento que se convierta en par motor. La diferencia entre ambas velocidades angulares se conoce como *deslizamiento* (slip).

Las ecuaciones de voltaje y flujo que describen el comportamiento de un motor de inducción dependen del tiempo. Esto introduce una complejidad para la resolución de estas ecuaciones por lo que se realiza unos cambios de variables para eliminar las dependencias de los términos que dependen del tiempo. Lo que se realiza es el paso de un marco de referencias trifásico (abc) a un marco de referencia de bifásico ortogonal. Este nuevo marco de referencia bifásico ortogonal puede ser estacionario (marco $\alpha\beta$) o rotatorio (marco dq). En este trabajo, se va a utilizar el marco $\alpha\beta$, aunque se va a necesitar algunas variables en el marco dq.

Ahora debemos pasar el voltaje que se encuentra representado en un marco trifásico (marco abc) al marco bifásico ortogonal estacionario (marco $\alpha\beta$) que vamos a utilizar. Para ello, se utilizará la transformada de Clarke que nos permite pasar del marco de referencia abc al marco $\alpha\beta$.

$$\begin{pmatrix} X_\alpha \\ X_\beta \end{pmatrix} = \sqrt{\frac{2}{3}} \begin{pmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ 0 & \frac{\sqrt{3}}{2} & \frac{-\sqrt{3}}{2} \end{pmatrix} \begin{pmatrix} X_a \\ X_b \\ X_c \end{pmatrix} \quad (3.2)$$

Para pasar del marco $\alpha\beta$ al marco dq, basta con aplicar una rotación de ángulo θ_r . Esta rotación se conoce como la Transformada de Park:

$$\begin{pmatrix} X_d \\ X_q \end{pmatrix} = \begin{pmatrix} \cos(\theta_r) & \sin(\theta_r) \\ -\sin(\theta_r) & \cos(\theta_r) \end{pmatrix} \begin{pmatrix} X_\alpha \\ X_\beta \end{pmatrix} \quad (3.3)$$

El modelo dinámico de un motor de inducción puede ser descrito en el marco de referencia estacionario $\alpha\beta$ a partir de las siguientes ecuaciones diferenciales:

$$\frac{di_{s\alpha}}{dt} = -\left(\frac{R_s}{L_s\delta} + \frac{R_r L_m^2}{L_s L_r^2 \delta}\right) i_{s\alpha} + \frac{L_m}{L_s L_r \delta} \omega_r \lambda_{r\beta} + \frac{R_r L_m^2}{L_s L_r^2 \delta} \lambda_{r\alpha} + \frac{1}{L_s \delta} V_{s\alpha} \quad (3.4)$$

$$\frac{di_{s\beta}}{dt} = -\left(\frac{R_s}{L_s\delta} + \frac{R_r L_m^2}{L_s L_r^2 \delta}\right) i_{s\beta} + \frac{R_r L_m}{L_s L_r \delta} \lambda_{r\beta} - \frac{L_m}{L_s L_r \delta} \omega_r \lambda_{r\alpha} + \frac{1}{L_s \delta} V_{s\beta} \quad (3.5)$$

$$\frac{d\lambda_{r\alpha}}{dt} = \frac{R_r L_m}{L_r} i_{s\alpha} - \frac{R_r}{L_r} \lambda_{r\alpha} - \omega_r \lambda_{r\beta} \quad (3.6)$$

$$\frac{d\lambda_{r\beta}}{dt} = \frac{R_r L_m}{L_r} i_{s\beta} - \frac{R_r}{L_r} \lambda_{r\beta} + \omega_r \lambda_{r\alpha} \quad (3.7)$$

$$\frac{d\omega_r}{dt} = \frac{1}{J} (T_e - T_l - f_b \omega_r) \quad (3.8)$$

Sin embargo, también será necesaria la ecuación del par electromagnético:

$$T_e = k_t (\lambda_{rd} i_{sq} - \lambda_{rq} i_{sd}) \quad (3.9)$$

junto con el flujo magnético del rotor en el marco de referencia dq :

$$\frac{d\lambda_{rd}}{dt} = \frac{L_m}{T_r} i_{sd} - \frac{1}{T_r} \lambda_{rd} + (\omega_e - \omega_r) \lambda_{rq} \quad (3.10)$$

$$\frac{d\lambda_{rq}}{dt} = \frac{L_m}{T_r} i_{sq} - \frac{1}{T_r} \lambda_{rq} - (\omega_e - \omega_r) \lambda_{rd} \quad (3.11)$$

El significado de las variables, sus valores y unidades en las que se mide se muestran en 5.

3.3. Descripción general del método

En el método que se propone, usaremos diferentes clases de redes neuronales para diseñar un sistema de tiempo real que posea tanto componentes continuos como discretos. Se va a usar un diseño típico de red neuronal que se basa en el entrenamiento, prueba y validación. Los valores de salida que generará el modelo del motor de inducción previamente creado en Simulink, que representan los aspectos dinámicos y funcionales del modelo ciberfísico que queremos controlar, son los datos de entrenamiento.

Como hemos mencionado anteriormente, nuestros datos de entrenamiento serán recogidos mediante la simulación de un modelo de motor de inducción ya que no disponemos de un motor de inducción físico para poder obtener estos datos a través de la observación y medición directa. Al hacerlo por simulación evitamos las posibles ineficiencias o inconsistencias que se puedan producir por algún error humano, podemos poner un tiempo de simulación para que cada vez que transcurra ese tiempo recibamos el valor de los parámetros, esto nos permite realizar innumerables pruebas de entrenamiento y cambios que de otra manera serían difíciles y muy costosos ya que

posiblemente no tendríamos disponibilidad total del sistema físico durante todo el tiempo necesario.

Sin embargo, no todo son ventajas al recopilar los datos por simulación. El principal y más obvio inconveniente es que por muy buena que sea la simulación, los datos siempre serán de una calidad inferior a los que podríamos haber obtenido por medición directa en un motor de inducción físico. Además, necesitamos recopilar los datos de entrenamiento muchas veces y eliminar los posibles valores anómalos de las series temporales para obtener un conjunto de datos que sea lo más realista posible.

Durante la ejecución del modelo del motor de inducción, las principales variables del sistema ciberfísico toman valores que serán usados para entrenar la red neuronal. Entre estas variables podemos destacar el voltaje en el marco de referencia abc (v_a, v_b, v_c), la corriente que circula por el estator en el marco $\alpha\beta$ (i_α, i_β), la velocidad sincrónica (ω_e) y el par de carga (T_l), entre otras.

El conjunto de herramientas Deep Learning Toolbox (DLT) de MATLAB nos permite generar bloques de Simulink que implementan las diferentes redes neuronales que se han creado (Feed-Forward, NARX, LRN). Los bloques de Simulink que implementan una red neuronal entrenada pueden sustituir a cualquier controlador PID en un sistema de control de lazo cerrado, que son los más usados en sistemas industriales para conseguir que la señal de salida se mantenga en el estado deseado, de acuerdo con la señal consigna, y producir una respuesta predecible del sistema incluso en situaciones en las que haya perturbaciones en las señales de entrada o en situaciones en que estas se presenten afectadas de ‘ruido’. Esta afirmación junto con las ventajas de utilizar redes neuronales para regular sistemas de control se pueden ver en [15].

Obtenemos los datos de entrenamiento tras simular nuestro modelo del motor de inducción en Simulink a partir del modelo publicado en [8], cuyo modelo del motor de inducción se puede descargar en la siguiente dirección: <https://lsi2.ugr.es/~mcapel/misclanea/motor/>.

El método propuesto comienza con un conjunto de datos formado con N puntos $\{v_\alpha, v_\beta, i_\alpha, i_\beta, \omega_r\}_{i=1}^N$, cuyos elementos individuales proporcionan información sobre propiedades estáticas en 2 dimensiones como el voltaje y la corriente o sobre propiedades dinámicas como la velocidad del rotor.

La nube de puntos o conjunto de datos (*dataset*, según la terminología al uso) que se va a utilizar para entrenar a las redes neuronales se puede obtener de diversas formas: midiendo directamente las variables en un motor de inducción real, obteniéndolas de una base de datos experimental de detección y diagnóstico de errores en el rotor en motores de inducción trifásicos o podemos obtenerla simulando un motor de inducción en Simulink, que es

la opción que se ha elegido en este caso. La obtención de los datos de entrenamiento debe repetirse una gran cantidad de veces y eliminarse los valores anómalos («outliers»), se realiza de esta forma ya que nuestro objetivo es obtener un conjunto de datos que sea lo más realista posible en la que la evolución de la velocidad del rotor (ω_r) y del flujo magnético que atraviesa la bobina del rotor ($\lambda_{r\alpha}, \lambda_{r\beta}$) a lo largo del tiempo pueda ser visualizado en el sistema ciberfísico. Otro objetivo que buscamos conseguir es predecir de manera precisa los valores anteriores y que deban poder reaccionar y estabilizarse frente a variaciones en el voltaje o en el par de carga. Para lograr esto, es de gran ayuda crear un modelo simulado del motor de inducción en Simulink.

Los datos de entrenamiento han sido tomados asumiendo una velocidad sincrónica $\omega_e = f \cdot \pi$ donde f es la frecuencia que alimenta al voltaje en la bobina del estator medida en Herzios ($f=60$ Hz). Por otra parte, el par de carga, T_l , toma valores entre $[-50, 50]$ durante toda la simulación y se mide en Newton/metro (Nm).

Los datos recogidos del modelo simulado son las corrientes y los voltajes en el marco de referencia bifásico estacionario conocido como $\alpha - \beta$. El tiempo de simulación fue de 250 segundos, los datos fueron tomados cada 0.005 segundos y el tamaño del conjunto de datos fue de 50000 tuplas.

El tipo de red neuronal que se va a usar es muy importante para conseguir una estimación precisa tanto de la velocidad del rotor como de los flujos del rotor del motor de inducción. Como se ha mencionado anteriormente, se ha usado el conjunto de herramientas DLT de MATLAB para implementar las diferentes redes neuronales. Los hiperparámetros relacionados con la estructura de las redes neuronales como el número de neuronas o el número de capas ocultas, entre otras, han sido escogidas a través de un proceso sistemático de prueba y error. Mientras que aquellos que están relacionados con el entrenamiento de la red como el número de épocas o la función de activación son dados por la función de entrenamiento escogida y se pueden ajustar manualmente.

3.3.1. Tipos de redes utilizadas en el estudio

En el capítulo anterior hemos mencionado la importancia que tiene escoger correctamente la clase de red neuronal para realizar la predicción. En esta sección vamos a explicar las tres clases de redes neuronales que se han escogido para la realización del trabajo. Se ha escogido una red neuronal no recurrente (FeedForward) y dos redes neuronales recurrentes (NARX y LRN).

3.3.1.1. Red neuronal FeedForward

Las redes neuronales FeedForward o, en español, redes neuronales prealimentadas son un tipo de redes neuronales artificiales donde entre las conexiones de las distintas capas no existe ningún ciclo. La información solo se mueve hacia adelante, desde los nodos de entradas, pasa por los nodos ocultos y llega a los nodos de salida.

Este tipo de red neuronal fue la primera y más sencilla forma de red neuronal artificial creada.

La estructura de este tipo de redes es la siguiente:

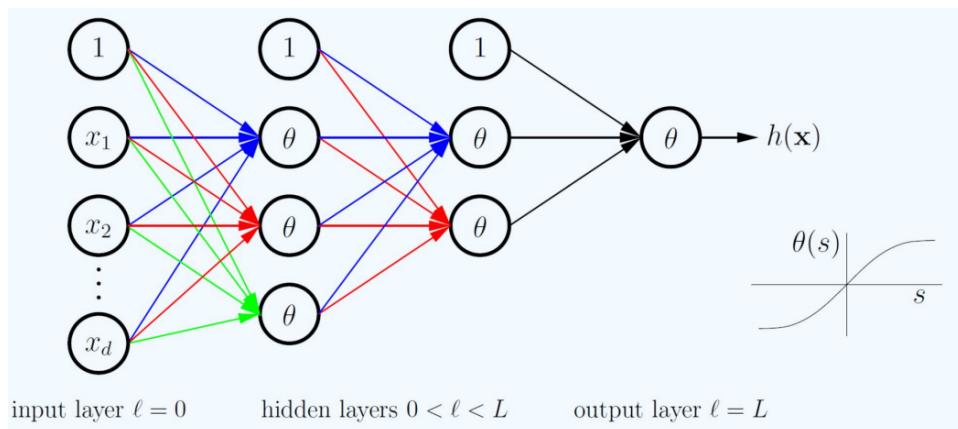


Figura 3.2: Estructura Red FeedForward. Imagen obtenida del Tema 8 de la asignatura Aprendizaje Automático.

En la figura podemos ver cómo de los nodos de entrada se propaga a las capas ocultas, se aplica la función de transferencia que se haya seleccionado (en el caso de la figura, la función de transferencia es una tangente hiperbólica), hasta llegar a la capa de salida que devuelve la función que haya estimado la red neuronal.

La función de transferencia o función de activación son funciones continuas, diferenciables, monotónicamente crecientes y se aplican a las entradas de cada neurona de la red de forma ponderada. A la salida de la neurona, estas funciones de activación, modifican el valor resultante o imponen un límite que hay que sobrepasar para poder llegar a la siguiente neurona.

Son muy importantes en las redes neuronales ya que tienen la capacidad de agregar no linealidad a la red. Si no hubiese funciones de activación, todas las redes neuronales podrían reducirse a un grupo de funciones lineales definidas por la entrada de la red, acotando la activación de las neuronas. Algunos ejemplos de funciones de activación son la función sigmoide o la función tangente hiperbólica.

En este tipo de redes, la técnica de aprendizaje que se utiliza es la Propagación hacia atrás o Backpropagation, que es la técnica más popular. Backpropagation es un algoritmo de aprendizaje automático que realiza una pasada hacia atrás para ajustar los parámetros del modelo, con el objetivo de minimizar el error cuadrático medio. Los valores de salida obtenidos de la red neuronal se van comparando con los valores reales de la variable a predecir para calcular el valor de alguna función de pérdida (en nuestro caso, ECM). El error es retroalimentado a través de la red y con esta información, el algoritmo va ajustando los pesos para reducir el valor de la función de error. Tras un número elevado de ciclos de entrenamiento, el valor de la red irá convergiendo a algún estado donde el valor de los errores será pequeño. Para ajustar los pesos correctamente, se usa el método de optimización no lineal conocido como *gradiente descendente*. El método gradiente descendente es una técnica de optimización iterativa que alcanza un óptimo local de una función siguiendo la dirección negativa del vector gradiente en cada punto y se utiliza habitualmente para el entrenamiento de modelos de aprendizaje automático y de redes neuronales [21].

3.3.1.2. Red neuronal NARX

Las redes NARX o redes autorregresivas no lineales con entradas exógenas son un tipo de red dinámica recurrente con conexiones de retroalimentación entre diferentes capas de la red. La ecuación que define el modelo NARX es:

$$y_t = f(y_{t-1}, y_{t-2}, \dots, u_{t-1}, u_{t-2}, \dots) \quad (3.12)$$

donde y es la variable a estudiar o de interés y u es la variable externa (o *exógena*). El siguiente valor de y_t viene dado tanto por los valores previos de la señal de salida ($\{y_{prev(t)}\}$) como por los valores previos de la señal de entrada ($\{u_{prev(t)}\}$) independiente.

Las redes NARX tienen múltiples aplicaciones. Se pueden utilizar como un predictor para predecir el siguiente valor de la señal de entrada, para el filtrado no lineal cuando se quiere que la salida sea una versión de la señal de entrada pero sin ruido y, una de las más importantes y por lo que ha sido escogida en este estudio, para el modelado de sistemas no lineales como es el motor de inducción. Es necesario explicar la configuración que se ha usado para el entrenamiento de la red. Se considera que la salida de la red NARX es una estimación de la salida (en nuestro caso, la velocidad del rotor) de un sistema dinámico no lineal (motor de inducción). La salida que se obtiene se utiliza también como entrada de la red como podemos ver:

Por otra parte, como el valor real de la salida está disponible, a partir del conjunto de datos que se ha utilizado para entrenar la red, durante toda la fase del entrenamiento de la red, se puede crear una estructura paralela de

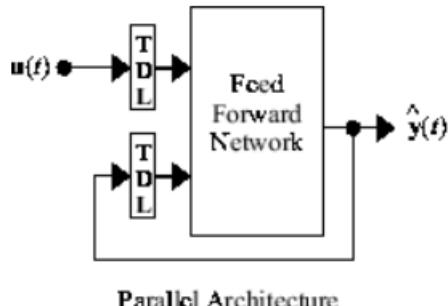
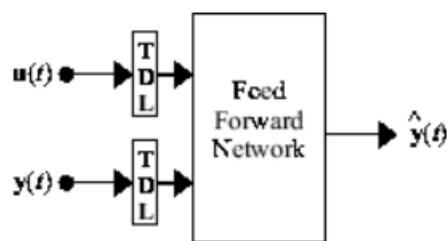


Figura 3.3: Arquitectura paralela de una red NARX. Imagen obtenida de <https://es.mathworks.com/help/deeplearning/ug/design-time-series-narx-feedback-neural-networks.html>

serie (una red con arquitectura secuencial pero alimentada por dos señales en paralelo) [16] en la que se utiliza la salida real en vez de la salida estimada como dato de entrada. La estructura de la red NARX sería:



Series-Parallel Architecture

Figura 3.4: Arquitectura paralela de serie de una red NARX. Imagen obtenida de <https://es.mathworks.com/help/deeplearning/ug/design-time-series-narx-feedback-neural-networks.html>

De esta forma conseguimos una predicción más fiable al usar el valor verdadero del valor a predecir y conseguimos una red totalmente prealimentada.

En este trabajo, se ha usado la red NARX con una arquitectura paralela de serie en la fase de entrenamiento para conseguir una predicción más fiable, y una vez entrenada, cambiamos la arquitectura paralela de serie de la red NARX a una arquitectura totalmente paralela. Es decir, la convertimos nuevamente en una red recurrente con el lazo de retroalimentación, como se ve en la figura 3.3, para realizar las predicciones.

3.3.1.3. Red neuronal LRN

Las redes neuronales LRN o Red Recurrentes de Capas son un tipo de red neuronal dinámica recurrente que contiene conexiones de retroalimentación con un único retardo en cada capa de la red neuronal a excepción de la última capa. Es decir, si tenemos una red LRN de N capas, las conexiones de retroalimentación ocurrirán en las primeras $N - 1$ capas. Esto permite a la red tener una respuesta dinámica no limitada a los datos de entrada de series de tiempo.

Podemos ver la estructura que posee una red LRN de 2 capas en la siguiente figura:

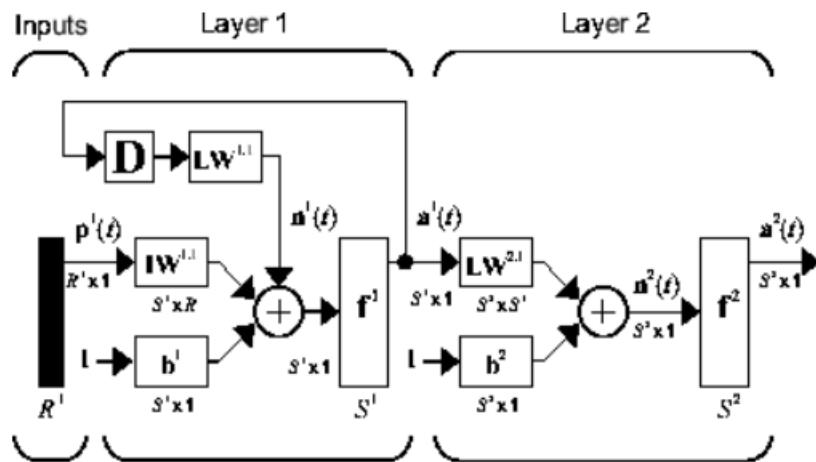


Figura 3.5: Estructura de una red LRN con 2 capas. Imagen obtenida de <https://es.mathworks.com/help/deeplearning/ug/design-layer-recurrent-neural-networks.html>

A diferencia de las redes neuronales FeedForward, las cuales no existe ningún ciclo entre las diferentes capas, y de las redes NARX, que utilizan como dato de entrada a la red el dato obtenido en la última capa, podemos apreciar que en las redes LRN la salida en cada capa (menos en la última capa) se utiliza como uno de los datos de entrada en esa misma capa. En la figura observamos como la salida obtenida en la capa 1 se usa junto a los datos de entrada para el cálculo al aplicarle la función de transferencia y obtener el valor de salida de la capa 1.

Las redes LRN se usan, mayoritariamente, en el filtrado y modelado de aplicaciones. Por ejemplo, uno de sus usos [12], es en la clasificación de imágenes donde estas redes LRN se combinan con redes neuronales convolucionales (CNN) preentrenadas y en la segmentación semántica, que es un algoritmo que asocia una etiqueta a cada píxel, en las que estas redes se combinan con redes VGG-16 para mejorar la tarea de la segmentación de

imágenes.

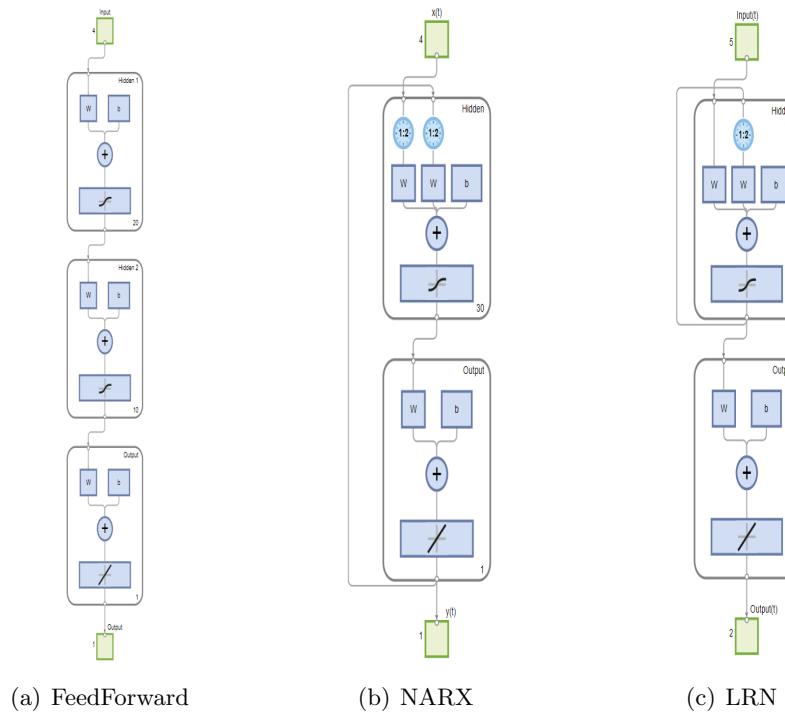


Figura 3.6: Arquitectura de las redes neuronales que se usarán en el estudio: **(a)** Red neuronal FeedForward con 2 capas ocultas de 20 y 10 neuronas. **(b)** Red neuronal NARX implementada como una red neuronal FeedForward de dos capas y con 30 neuronas. **(c)** Red neuronal LRN implementada como una red neuronal FeedForward de dos capas y 11 neuronas.

3.3.2. Funciones de entrenamiento escogidas

La función de entrenamiento que se ha usado en tres redes neuronales, en dos FeedForward y en la red NARX, ha sido el de Levenberg-Marquardt (`<trainlm>`) y en la otra red neuronal, LRN, se ha usado la función de Regularización Bayesiana (`<trainbr>`) y la función objetivo que se ha usado ha sido el error cuadrático medio (ECM).

3.3.2.1. Algoritmo Levenberg-Marquardt

El algoritmo de Levenberg-Marquardt se utiliza para resolver problemas de mínimos cuadrados no lineales. Este algoritmo es una combinación de los siguientes dos métodos: gradiente descendente y Gauss-Newton. Es muy utilizado para el entrenamiento de redes neuronales ya que estas utilizan

como índice de rendimiento el Error Cuadrático Medio. Partamos del método de Newton [3] para optimizar una función de rendimiento $\text{ECM}(x) = F(x)$, la ecuación es:

$$x_{k+1} = x_k - A_k^{-1} g_k, \quad (3.13)$$

donde A_k^{-1} es la matriz Hessiana de $F(x)$ y g_k es la matriz Jacobiana de $F(x)$.

El método de Gauss-Newton se diseñó para conseguir aproximarse a la velocidad de entrenamiento de segundo orden sin la necesidad de calcular la matriz Hessiana. Sea un campo escalar $f : \mathbb{R}^n \rightarrow \mathbb{R}$, entonces la matriz hessiana es una matriz de tamaño n que tiene como entradas las derivadas parciales de segundo orden:

$$H(f) = \begin{pmatrix} \frac{\partial f}{\partial x_1^2}(x) & \frac{\partial f}{\partial x_1 \partial x_2}(x) & \dots & \frac{\partial f}{\partial x_1 \partial x_n}(x) \\ \frac{\partial f}{\partial x_2 \partial x_1}(x) & \frac{\partial f}{\partial x_2^2}(x) & \dots & \frac{\partial f}{\partial x_2 \partial x_n}(x) \\ \dots & \dots & \dots & \dots \\ \frac{\partial f}{\partial x_n \partial x_1}(x) & \frac{\partial f}{\partial x_n \partial x_2}(x) & \dots & \frac{\partial f}{\partial x_n^2}(x) \end{pmatrix} \quad (3.14)$$

Como la función de rendimiento es ECM que es una suma de cuadrados, es posible aproximar la matriz Hessiana de la siguiente manera:

$$H(x) = J^T(x)J(x) \quad (3.15)$$

donde J es la matriz Jacobiana. La matriz Jacobiana de una función vectorial es una matriz cuyas entradas son las derivadas de primer orden de la función. Sea un campo vectorial $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ y denotemos f_1, \dots, f_m sus componentes escalares, entonces su matriz jacobiana será la siguiente matriz de tamaño $m \times n$:

$$J(f) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \dots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \dots & \frac{\partial f_2}{\partial x_n}(x) \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_m}{\partial x_1}(x) & \frac{\partial f_m}{\partial x_2}(x) & \dots & \frac{\partial f_m}{\partial x_n}(x) \end{pmatrix} = \begin{pmatrix} \nabla f_1(x) \\ \nabla f_2(x) \\ \vdots \\ \nabla f_m(x) \end{pmatrix} \quad (3.16)$$

donde ∇f_i es el gradiente de la función sobre la variable i -ésima.

El gradiente de la función de rendimiento, en este caso, lo podemos calcular como:

$$\nabla F(x) = J^T(x)e(x) \quad (3.17)$$

donde $e(x)$ es el vector de errores.

Tras todas estas explicaciones y aproximaciones, si sustituimos en la ecuación que describe el método de Newton, las nuevas aproximaciones de la matriz Jacobiana y Hessiana llegamos a:

$$x_{k+1} = x_k - (J^T(x_k)J(x_k))^{-1}J^T(x_k)e(x_k) \quad (3.18)$$

El único problema del método de Gauss-Newton es que la matriz Hessiana puede no ser invertible. Sin embargo, podemos superar este impedimento si modificamos la aproximación que hacemos sobre la Hessiana de la siguiente forma:

$$G = H + \mu I = J^T J + \mu I \quad (3.19)$$

Es fácil ver que ahora la matriz sí es invertible. Suponemos que los valores propios de H son $\{\lambda_1, \dots, \lambda_n\}$ y sus vectores propios son $\{z_1, \dots, z_n\}$. Entonces:

$$Gz_i = (H + \mu I)z_i = Hz_i + \mu z_i = \lambda_i z_i + \mu z_i = (\lambda_i + \mu)z_i \quad (3.20)$$

Los vectores propios de G y H son los mismos y los valores propios de G son $\lambda_i + \mu$. Podemos hacer que la matriz G sea definida positiva si incrementamos el valor de μ hasta que $\lambda_i + \mu > 0$ para todo i , y de esta forma, la matriz será invertible.

Así que, finalmente, el algoritmo de Levenberg-Marquardt para actualizar los pesos es el siguiente:

$$x_{k+1} = x_k - (J^T(x_k)J(x_k) - \mu_k I)^{-1}J^T(x_k)e(x_k) \quad (3.21)$$

Las ventajas de este algoritmo es que al principio utiliza la técnica de gradiente descendente para minimizar más rápido en las primeras iteraciones. Cuando se llega a un mínimo local, el parámetro μ_k será cero y a partir de ahí, se utiliza la técnica de Gauss-Newton para el resto de la optimización. Como se dijo anteriormente, todo esto ha hecho que sea uno de los métodos más usado en el entrenamiento de redes neuronales.

3.3.2.2. Regularización Bayesiana

La regularización bayesiana pretende resolver problemas de generalización y optimización de pesos ya que presenta una convergencia a un conjunto de pesos óptimos los cuales toman los datos iniciales como una distribución de probabilidad y de éstos extrae información necesaria para generar un conjunto de parámetros denominados hiperparámetros que representen la distribución de los datos de entrada [6]. Más concretamente, en el entrenamiento de redes neuronales, minimiza una combinación lineal de errores cuadráticos y pesos. Como función de rendimiento vamos a usar $ECM = E_D$. E_D será la suma de los errores al cuadrado. Normalmente, el entrenamiento tiene el objetivo de reducir esta suma (E_D). Sin embargo, la regularización añade un nuevo término. La función objetivo pasa a ser: $F = \alpha E_W + \beta E_D$ donde E_W es la suma al cuadrado de los pesos de la red neuronal y α y β son

parámetros de la función objetivo. La diferencia de los valores α y β marcan el aprendizaje. Si α es mucho mayor que β , el entrenamiento se centrará en la reducción del tamaño de los pesos a expensas de los errores en la red mientras que si β es mucho mayor que α , el entrenamiento se centrará en reducir los errores. Los pasos necesarios para la regularización bayesiana de los parámetros, usando el método de Gauss-Newton, son los siguientes [10]:

1. Se inicializan las funciones objetivos α , β y los pesos. Tomamos los valores $\alpha = 0$, $\beta = 0$ y se usa el método de Nguyen-Widrow para inicializar los pesos.
2. Se aplica una iteración del algoritmo Levenberg-Marquardt para minimizar la función $F(w) = \alpha E_W + \beta E_D$.
3. Se calcula el número de parámetros efectivos $\gamma = N - 2\alpha \cdot \text{traza}((H)^{-1})$, donde H es la aproximación a la matriz Hessiana que viene dada por $H = 2\beta J^T J + 2\alpha I_N$.
4. Se calculan nuevos estimadores para las funciones objetivos $\alpha = \frac{\gamma}{2E_W(w)}$ y $\beta = \frac{n-\gamma}{2E_D(w)}$.
5. Se iteran los pasos 2-3-4 hasta que se obtenga una convergencia en los pesos.

Por supuesto, el número de capas ocultas y el número de neuronas en cada capa también se puede ajustar para obtener el máximo rendimiento posible.

3.3.3. Entrenamiento de la Red Neuronal

Los hiperparámetros de una red neuronal son variables de configuración externa que se usan para administrar el entrenamiento de modelos de aprendizaje automático y se configuran manualmente antes de entrenar al modelo. Estos hiperparámetros determinan características claves en el modelo, como su arquitectura, su complejidad o su tasa de aprendizaje.

Se destacan dos grandes grupos de hiperparámetros en una red neuronal, los hiperparámetros relacionados con la arquitectura de la red:

- Número de neuronas que son el número de nodos que posee cada capa de la red. Debido a la baja interpretabilidad de las redes neuronales, hay que realizar ensayos de prueba y error para conseguir los mejores resultados.
- Número de capas de la red son la parte de la red neuronal que contiene a las neuronas. Al igual que con el número de neuronas, los mejores resultados se consiguen con ensayos de prueba y error.

- «Dropout» o dilución es una técnica de regularización para reducir el sobraejueste en redes neuronales artificiales y consiste en la omisión aleatoria de neuronas durante la fase de entrenamiento. Se destacan dos casos: dilución débil, se eliminan pocas conexiones, y dilución fuerte, se eliminan muchas conexiones. Su uso tiene importantes implicaciones ya que, al usar dilución, se añade ruido.
- Distintos esquemas de inicialización de los pesos. A la hora de inicializar los pesos, pueden tomar diferentes valores. Los más utilizados son inicializar los pesos a cero, a uno, al valor de una constante o utilizar una distribución normal o uniforme. Este hiperparámetro no es muy determinante a la hora del entrenamiento pero es muy importante para hacer que nuestra red converja a un mínimo adecuado porque a veces puede hacer que la red no empiece a entrenar porque no consigue converger.
- La función de activación como ya se explicó en 3.3.1.1.

y los relacionados con el entrenamiento:

- Tasa de aprendizaje o «learning rate» que es un factor que se usa en la actualización de los pesos. Este valor indica la importancia que se le da al error al actualizar cada peso, es decir, cómo de abrupto son los cambios en los pesos. Si se le otorga un valor demasiado alto, el cambio de los pesos será muy grande de una iteración a otra y puede que salte el mínimo al que se debe llegar o que no converja, mientras si su valor es demasiado pequeño puede que no se llegue al mínimo.
- «Momentum» es una técnica de suavizado para evitar oscilaciones excesivas en la dirección del gradiente y aumentar la velocidad de convergencia. En vez de actualizar los pesos usando la información del gradiente actual, utiliza un promedio ponderado del gradiente actual y de los gradientes anteriores para calcular la dirección de actualización. Este algoritmo requiere dos parámetros: el coeficiente o tasa de aprendizaje y el coeficiente de momentum. Por esta razón, hay que tener las mismas consideraciones que con la tasa de aprendizaje.
- Número de épocas que son el número de veces que cada ejemplo de entrenamiento va a pasar por la red. Si el número de épocas que establecemos es alto puede provocar sobreajuste («overfitting») porque el peso y el sesgo en una red neuronal converge para una cierta época, y a partir de esa época, se deja de aprender y es un gasto de tiempo innecesario. Por otra parte, si es bajo puede provocar «underfitting» (infrajuste) y que no aprenda lo suficiente.

- Tamaño de los lotes o «batch size» indican el número de muestras que se utilizan para actualizar los pesos. Este hiperparámetro no es tan crítico ya que se suele establecer su valor como una potencia de dos. Si es demasiado pequeño, entrena más rápido pero no aprende las características más significativas y si el demasiado grande tiene en cuenta los casos más importantes pero entrena más lento.

El principal objetivo al entrenar una red neuronal es minimizar la función objetivo $F(x, \Pi_i)$ de manera iterativa mientras ajustamos el conjunto de hiperparámetros $\Pi_i = \alpha, \beta, \gamma, \dots$ respecto a cada una de las K muestras de entrenamiento, que previamente se habían escogido como un subconjunto de la nube de puntos, por ejemplo,

$$x = \{v_\alpha^i, v_\beta^i, i_\alpha^i, i_\beta^i, \omega_r^i\}_{i=1}^K \quad (3.22)$$

Este proceso se puede expresar a través de la siguiente ecuación:

$$F(x, \Pi_i) = l.r.(\{v_\alpha^i, v_\beta^i, i_\alpha^i, i_\beta^i, \omega_r^i\}_{i=1}^K, \Pi_i) + r(\Pi_i) \quad (3.23)$$

donde F es la función objetivo, $l.r.$ representa la relación lineal que existe entre cada conjunto de entrenamiento con los hiperparámetros Π_i y la función $r()$ representa la penalización o errores que se podrían cometer en los valores de cada conjunto de hiperparámetros.

La optimización de la función objetivo se puede llevar a cabo a través de diferentes técnicas como aprendizaje por refuerzo, heurísticas, gradiente descendente... Como ya se ha mencionado, hemos optado por la técnica de gradiente descendente ya que las funciones que describen el modelo cibérico de un motor de inducción son diferenciables [8]. Como F conserva sus propiedades, si utilizamos gradiente descendente para el cálculo de $\min_{\Pi_i} F(x, \Pi_i)$ no va a diverger.

El entrenamiento de cada red neuronal ha sido realizado de manera «offline» [2, 8], usando los datos de voltaje y corrientes obtenido durante la ejecución del modelo para ahorrar tiempo de implementación. La velocidad del rotor fue predicha primeramente por una red neuronal Feed-Forward y luego por una red NARX mientras que los flujos del rotor fueron predichos por una red neuronal Feed-Forward y después por una red LRN.

3.3.4. Validación de la Red Neuronal

Para obtener una función objetivo útil que se adapte a nuestro objetivo, definiremos una función de coste que optimice y complete el rendimiento del modelo que vamos a evaluar, el cual hemos escogido basándonos en el Error

Cuadrático Medio sobre cuatro parámetros: voltajes (v_x), corrientes (i_x), velocidad del rotor (ω_r) y flujos del rotor (λ_{rx}) que afectan a la fiabilidad y eficiencia de nuestro modelo.

Se ha usado como función de rendimiento el Error Cuadrático Medio (ECM) porque es un estimador que mide el promedio de los errores al cuadrado, es decir, la diferencia entre el valor real o esperado y el valor predicho u obtenido. Además, al ser el Error Cuadrático Medio un momento de segundo orden centrado por lo que incorpora tanto la varianza del estimador como su sesgo. El ECM se calcula de la siguiente manera. Sea \hat{Y} un vector de tamaño n con los valores predichos y sea Y el vector de tamaño n con los valores esperados, entonces:

$$\text{ECM} = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2 \quad (3.24)$$

Podemos intercalar la dimensión del modelo para obtener una evaluación multidimensional del rendimiento ya que ECM puede guiar las iteraciones durante la aplicación del gradiente descendente,

$$I_{k_x} = \sqrt{\frac{\sum_{j=1}^K ((\vec{i}_{x_j} - \vec{i}_{x_j}^{ref})^2)}{K}} \quad (3.25)$$

$$V_{k_x} = \sqrt{\frac{\sum_{i=1}^K ((\vec{v}_{x_i} - \vec{v}_{x_i}^{ref})^2)}{K}} \quad (3.26)$$

$$W_{r_k} = \sqrt{\frac{\sum_{i=1}^K ((\omega_{r_i} - \omega_{r_i}^{ref})^2)}{K}} \quad (3.27)$$

$$L_{r_{k_x}} = \sqrt{\frac{\sum_{i=1}^K ((\vec{\lambda}_{r_{x_i}} - \vec{\lambda}_{r_{x_i}}^{ref})^2)}{K}} \quad (3.28)$$

Donde el valor de referencia de las variables que se usan para el cálculo de ECM son los valores de voltaje, corriente, velocidad del rotor y flujos del rotor medidos en cada uno de las K muestras de entrenamiento. Podemos encontrar el modelo óptimo minimizando de la siguiente manera:

$$F(\Pi) = \log \alpha \cdot I_{k_x} + \beta \cdot V_{k_x} + \gamma \cdot W_{r_k} + \delta \cdot L_{r_{k_x}}, \quad (3.29)$$

donde se ha aplicado el logaritmo a la función para minimizarla. Hemos aplicado el logaritmo a la función porque en los métodos que involucran al

gradiente, suele funcionar mejor optimizando $\log f(x)$ que $f(x)$, ya que el gradiente de $\log(f(x))$ escala mejor al reducir el rango de valores en los que se mueve la función. Es decir, ahora los datos tienen un tamaño que refleja de forma consistente y útil la geometría de la función, facilitando la selección de un tamaño de paso adecuado y llegando al óptimo en menos pasos. Además, el logaritmo es una función monótona y al aplicarlo conseguimos una forma numéricamente estable.

En esta ecuación, 3.29, α , β , γ y δ son los pesos de los hiperparámetros y I_{k_x} , V_{k_x} , W_{r_k} y $L_{r_{k_x}}$ representan los factores de rendimientos de la corriente de entrada, voltaje de entrada y de la velocidad de rotor y flujos del rotor que se han definido en 3.25, 3.26, 3.27 y 3.28.

3.3.5. Exportando el Bloque a Simulink

Tras comprobar la eficiencia y precisión de la red neuronal que se ha generado, el siguiente paso es exportarla como un bloque Simulink utilizando el comando «gensim(nombre_red)» que nos proporciona DLT. Los pasos para obtener un estimador preciso para la velocidad del rotor como para el flujo del rotor en el marco de referencia $\alpha - \beta$ son los marcados a continuación. Este código muestra que se ha seleccionado una red neuronal de tipo Feed-Forward para obtener estas predicciones y que tiene 2 capas ocultas con 10 y 5 neuronas cada una:

1. Se implementa un modelo del motor de inducción en Simulink.
2. Se ejecuta la simulación del modelo implementado en Simulink.
3. Se recoge los datos de entrada (voltajes, corrientes, velocidad de rotor).
4. Se recoge los datos de salida (velocidad de rotor y flujos del rotor en $\alpha - \beta$).
5. Se diseña la red neuronal Feed-Forward con las siguientes instrucciones:
 - a) `ff_net = feedforwardnet([20,10]);`
 - b) `ff_net.trainFcn='trainlm';`
 - c) `ff_net.trainFcn='trainlm';`
 - d) `ff_net.divideFcn='dividetrain';`
 - e) `ff_net.divideMode='sample';`
 - f) `ff_net.trainParam.EPOCHS=500;`

La parte principal en el diseño de una red neuronal es escoger el tipo de red neuronal que se va a usar. Con la instrucción *feedforwardnet([20,10])*, se especifica que se quiere utilizar una red neuronal FeedForward con dos capas ocultas y que en la primera capa haya 20 neuronas y en la segunda capa haya 10 neuronas. Tras esto, se indica, con la instrucción *trainFcn*, qué función de entrenamiento se va a usar en cada capa de la red neuronal. Para esta red FeedForward, en cada capa se usará el algoritmo de Levenberg-Marquardt. Con *divideFcn* se establece la división de los datos en tres conjuntos (entrenamiento, validación y test) y con *divideMode* se define cómo se dividirá la dimensión de los datos objetivos cuando se llama a la función de división de los datos, por defecto su valor es 'sample'. Finalmente, con *trainParam.EPOCHS* se indica el número máximo de épocas con las que se va a entrenar a la red neuronal.

6. Se configura y entrena la red neuronal de la siguiente forma:

- a) `ff_net=configure(ff_net,datos_entrada,datos_salida);`
- b) `[ff_net,tr]=train(ff_net,datos_entrada,datos_salida);`

Capítulo 4

Análisis y discusión de los resultados

4.1. Redes neuronales para estimar la velocidad del rotor y el flujo magnético del rotor

Los pasos que se han seguido para conseguir unos estimadores precisos para la velocidad del rotor (ω_r) y para el flujo magnético del rotor (λ_α y λ_β) son los siguientes:

1. Se implementa un modelo del motor de inducción en Simulink.
2. Se ejecuta el modelo en Simulink.
3. Se recogen los datos de entrada.
4. Se recogen los datos de salida.
5. Se diseña y entrena la red neuronal.

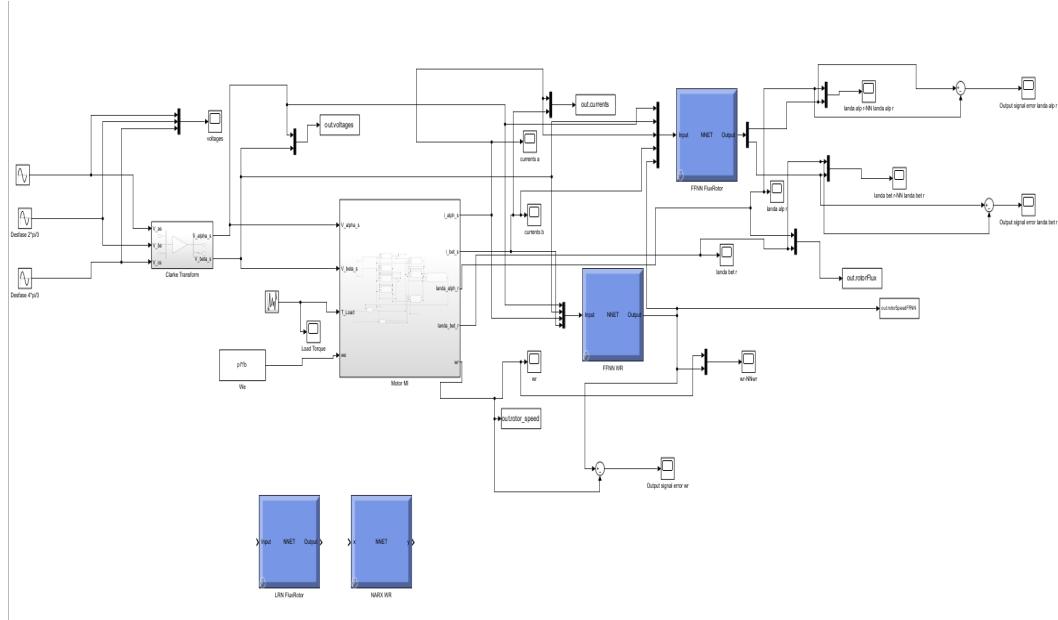


Figura 4.1: Modelo del motor de Inducción en Simulink.

En el motor podemos observar seis bloques importantes. Hay cuatro bloques de color azul que se corresponden con el bloque Simulink que se corresponde con las 4 redes neuronales que se han creado (dos redes neuronales FeedForward, una red NARX y una red LRN). En la figura, los bloques que están conectados al modelo son los bloques correspondientes a las dos redes FeedForward.

Por otra parte, observamos un pequeño bloque gris en la zona de la izquierda y este bloque ha sido creado para pasar el voltaje que viene dado de forma trifásica (v_a, v_b, v_c) a un marco de referencia bifásico estacionario (v_α, v_β) a través de la Transformada de Clarke. El bloque más grande que se encuentra en el centro de la figura, corresponde con el motor de inducción. A partir del voltaje en el marco $\alpha\beta$, de la velocidad de sincronismo (ω_e) y del par de carga (T_l), obtenemos tanto la tensión como los flujos magnéticos del rotor en el marco $\alpha\beta$ y también la velocidad del rotor (ω_r).

Los datos de entrada, los datos de salida y el diseño y entrenamiento de las redes neuronales son diferentes según el parámetro a estimar y la clase de red neuronal escogida.

Los datos han sido recogidos asumiendo un par de carga aleatorio. En este caso, el par de carga ha tomado valores entre $[-50, 50]$ durante los 250 segundos que ha durado la simulación. Así ha evolucionado el par de carga:

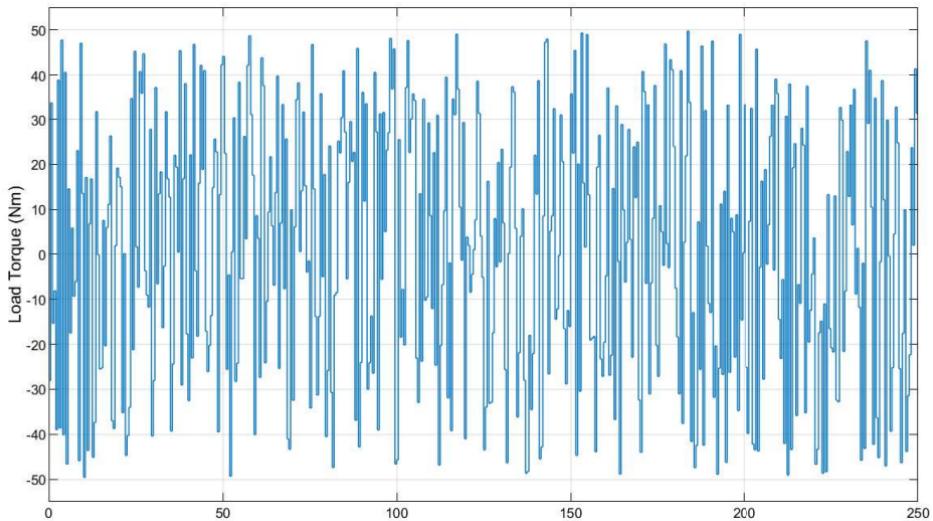


Figura 4.2: Valores del par de carga durante la simulación del modelo.

No se ha mantenido constante el par de carga porque se quería comprobar cómo estimaban las redes neuronales con los diferentes cambios de valores que sufría el par de carga. Estas perturbaciones en el par de carga, influirá en la velocidad del rotor.

4.1.1. Redes neuronales para estimar la velocidad del rotor

Los valores de entrada que se han usado para entrenar las redes neuronales (FFNN y NARX) que van a estimar la velocidad del rotor han sido: el voltaje (v_α, v_β) medido en el marco de referencia $\alpha\beta$ y la corriente (i_α, i_β), también medida en el marco $\alpha\beta$. Obviamente, el valor de salida ha sido la velocidad del rotor (ω_r). Estos conjuntos de datos han sido obtenidos tras simular el modelo del motor de inducción en Simulink.

4.1.1.1. FeedForward para la velocidad del rotor

Para el diseño, entrenamiento, test y validación de la red FeedForward se ha usado la aplicación MATLAB junto al conjunto de herramientas DLT.

Los siguientes pasos muestran cómo se ha creado y diseñado la red FeedForward para estimar la velocidad del rotor:

- `ff_net = feedforwardnet([20,10]);`
- `ff_net.trainFcn='trainlm';`

- ff_net.divideFcn='dividetrain';
- ff_net.divideMode='sample';
- ff_net.trainParam.EPOCHS=500;
- ff_net=configure(ff_net,transpose(InputDataWR),transpose(RSTT.Data));
- [ff_net,tr]=train(ff_net,transpose(InputDataWR),transpose(RSTT.Data));
- gensim(ff_net);

De esta forma, se crea una red neuronal FeedForward con dos capas ocultas y que contienen 20 neuronas y 10 neuronas, respectivamente; se usa como función de entrenamiento el algoritmo de Levenberg-Marquardt, se realiza un entrenamiento de, como máximo, 500 épocas; se configura y entrena con el conjunto de datos de entrenamiento InputDataWR, que contiene los voltajes y las tensiones en el marco $\alpha\beta$; y como datos a aprender, la velocidad de rotor que se encuentra almacenada en RSTT.Data. Finalmente, con el comando gensim(ff_net) se convierte la red neuronal en un bloque de Simulink (uno de los bloques azules que se observan en la figura del motor de inducción).

La estructura de la red neuronal se puede ver en la siguiente figura:

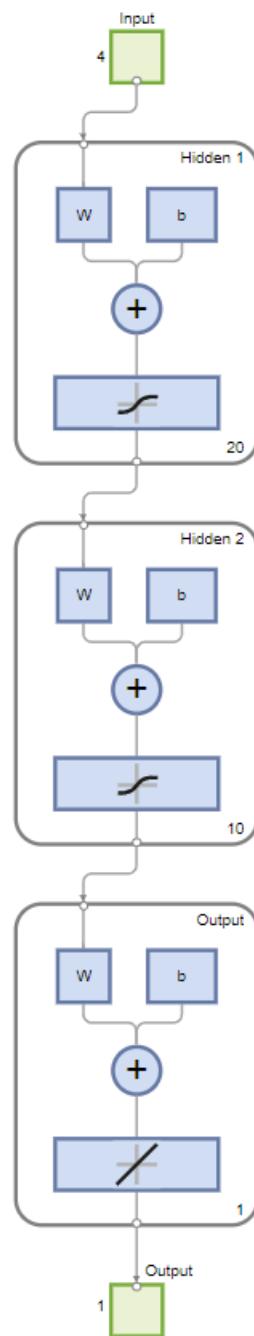


Figura 4.3: Estructura de la red neuronal FeedForward formada por 2 capas ocultas con 20 y 10 neuronas para estimar la velocidad del rotor.

La información que se ha obtenido tras el entrenamiento de la red fue la siguiente:

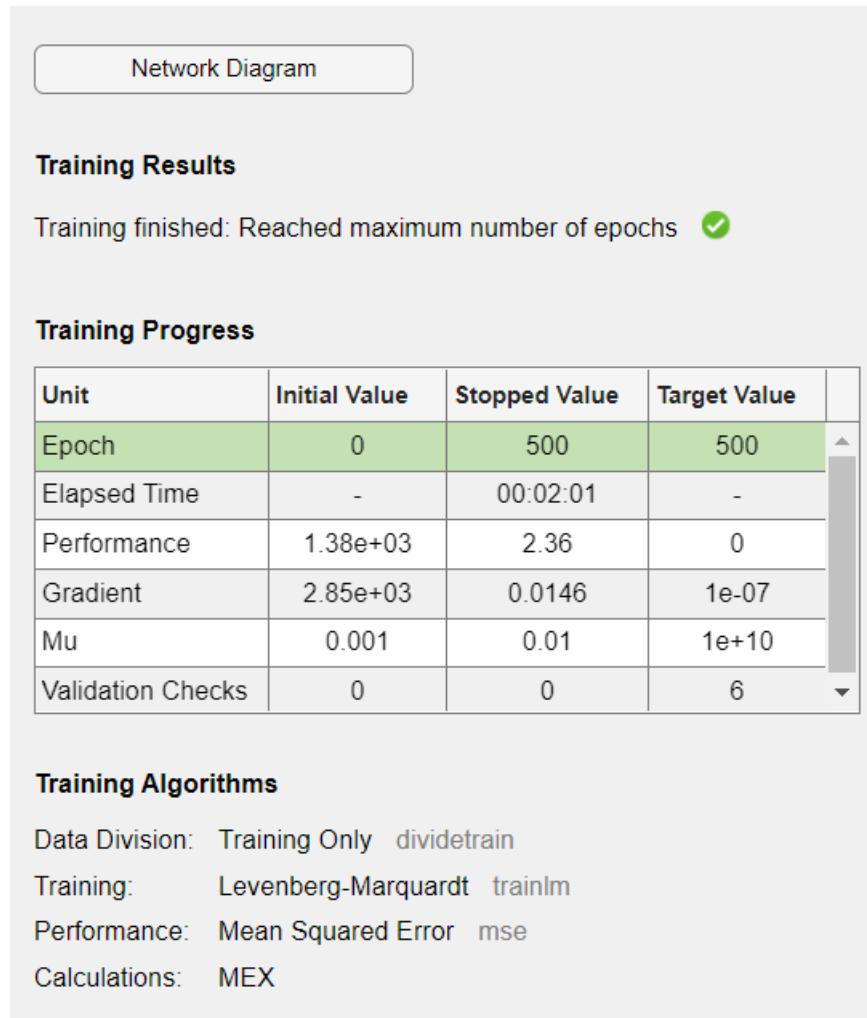


Figura 4.4: Resultados del entrenamiento de la red FeedForward para estimar la velocidad del rotor

Entre los resultados que nos muestra el entrenamiento podemos observar que el criterio de parada ha sido el haber realizado el número máximo de épocas, el valor del gradiente se encuentra en el orden de 10^{-2} (0.0146). Igualmente, el valor del rendimiento de la red es un número cercano a cero, por lo que podemos deducir que el entrenamiento ha sido bueno y, como también informará la figura 4.5, que la función para medir el rendimiento es el Error Cuadrático Medio.

El entrenamiento también nos mostró cómo evolucionó el rendimiento de la red según iban avanzando las épocas. también obtenemos un histograma de errores y un diagrama de regresión entre los valores estimados por la red FeedForward y los valores simulados por el motor.

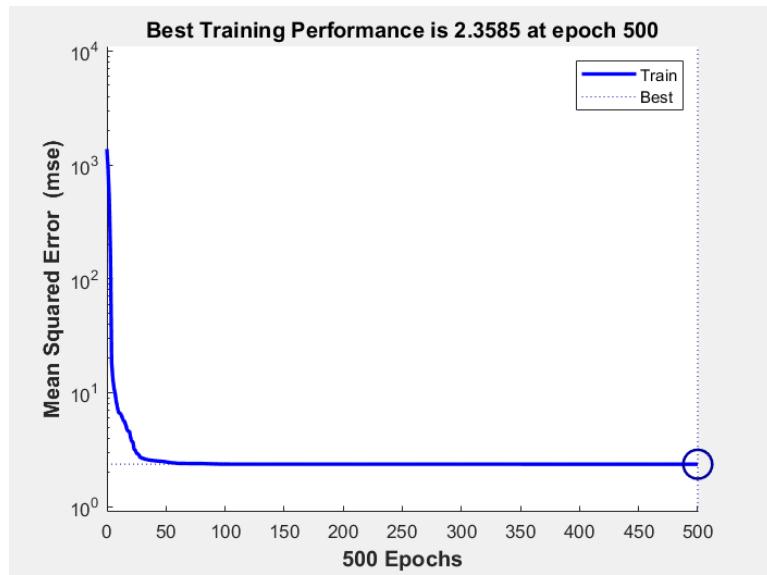


Figura 4.5: Rendimiento de la red FFNN para ω_r considerando ECM

Se aprecia una importante mejora en el rendimiento en las primeras 50 épocas ya que el valor baja de tener un orden de 10^3 a un valor cercano a 2 mientras que en las siguientes 450 épocas, el rendimiento sigue mejorando pero apenas se puede notar en la figura.

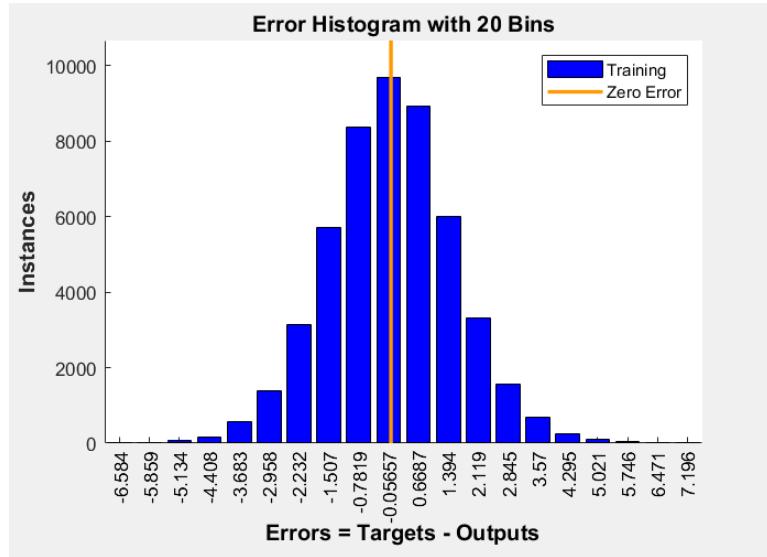


Figura 4.6: Histograma de Errores de FFNN sobre ω_r

En el histograma de errores podemos ver cómo la mayoría de los valores

predichos se diferencian de los valores reales en menos de 1.507 unidades y que la diferencia más repetida es de 0.05657 unidades. Podemos decir que los valores predichos por la red neuronal son bastante similares a los valores simulados por el motor.

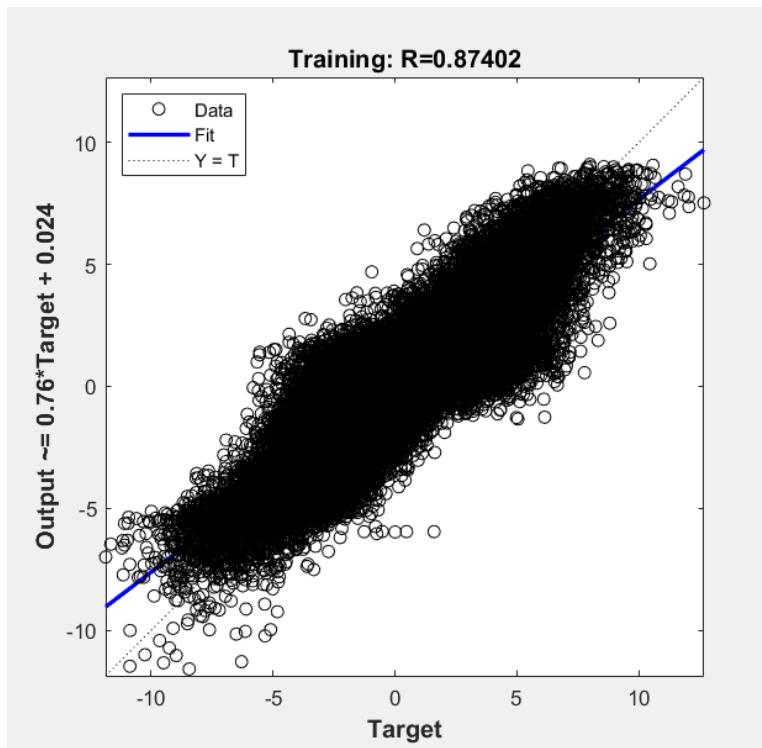


Figura 4.7: Regresión entre los valores estimados por la FFNN y los valores simulados por el motor.

En la figura de la regresión, aunque no se aprecia claramente debido a la gran cantidad de puntos que aparece en la gráfica, observamos que siguen una recta. Se nos aporta el valor del coeficiente de correlación que es $R = 0.87402$ por lo que podemos afirmar que existe una cierta dependencia lineal entre los valores de la velocidad del rotor estimados por la red FFNN y los valores de la velocidad del rotor simulados por el motor.

4.1.1.2. NARX para la velocidad del rotor

Al igual que para la red FeedForwaard, para realizar el diseño, entrenamiento, test y validación de la red NARX se ha usado la aplicación MATLAB junto con las herramientas de DLT.

Los siguientes pasos muestran cómo se ha creado y diseñado la red NARX

para estimar la velocidad del rotor:

- `delay1=[1:2];delay2=[1:2];`
- `narx_net = narxnet(delay1,delay2,30);`
- `narx_net.trainFcn='trainlm';`
- `narx_net.trainParam.min_grad=10-10;`
- `[x,xi,ai,t]=preparets(narx_net,W,{},T);`
- `narx_net.trainParam.EPOCHS=1000;`
- `narx_net.divideParam.trainRatio = 70/100;`
- `narx_net.divideParam.testRatio = 15/100;`
- `narx_net.divideParam.valRatio = 15/100;`
- `[narx_net,tr] = train(narx_net,x,t,xi,ai);`
- `narx_net_closed=closeloop(narx_net);`
- `gensim(narx_net_closed);`

De esta forma, se crea una red neuronal NARX con una capa oculta y que contienen 30 neuronas y se aplica un retardo [1:2] tanto a los datos de entrada como a los datos de retroalimentación.

Al aplicarle un retardo [1:2] tanto a los datos de entrada como a los datos de retroalimentación, nos estamos refiriendo a que usamos los dos valores anteriores para calcular el nuevo valor. Por ejemplo, usando la notación de 3.12, que para calcular cualquier valor, con $t \geq 3$, se realiza de la siguiente forma:

$$y_t = f(y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}) \quad (4.1)$$

Se usa como función de entrenamiento el algoritmo de Levenberg-Marquardt, y modificamos el valor mínimo del gradiente a 10^{-10} para que el entrenamiento se detenga si el gradiente alcanza ese valor y se indica que este se realice como máximo durante 1000 épocas. La instrucción `preparets()` simplifica la tarea de establecer el formato de las series de tiempo de entrada y salida, ya que es una tarea compleja y en la que se suelen cometer errores. Toma como parámetros la propia red NARX, `W` que son los datos de entrada (voltajes y tensiones medidos en el marco $\alpha\beta$) convertidos en un celda de forma de array para la red y `T` que contiene los datos de salida (ω_r) convertidos también en una celda de forma de array. Se ha dividido manualmente el conjunto de datos de entrenamiento en un 70 % para entrenamiento y un

15 % para validación y test, cada uno, debido a que las redes NARX tienden con más frecuencia llegar a situaciones de sobreajuste. Finalmente, se utiliza la función `closeloop` para cambiar la configuración de la red NARX de una arquitectura *serie*, para el entrenamiento, a una estructura paralela que es más útil para la predicción y que representa el comportamiento regular de la red NARX. La arquitectura serie-paralela solo se usa para el entrenamiento de la red. Por último, con el comando `gensim(narx_net_closed)` se convierte la red neuronal en un bloque de Simulink.

La estructura de la red neuronal se puede ver en la siguiente figura:

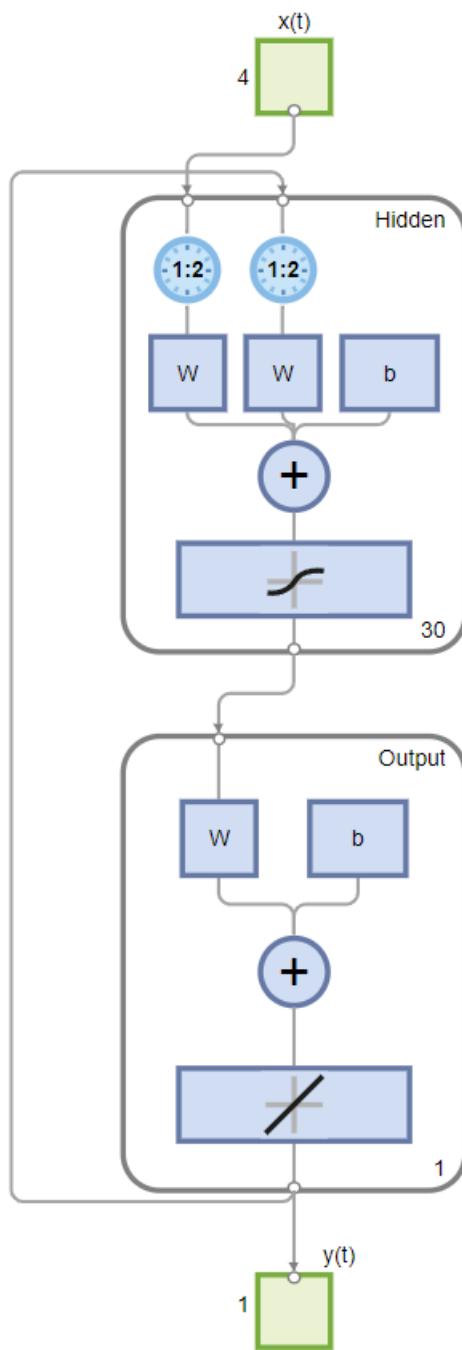


Figura 4.8: Estructura de la red neuronal NARX formada por una capa oculta con 30 neuronas para estimar la velocidad del rotor.

La información que se ha obtenido tras el entrenamiento de la red fue el siguiente:

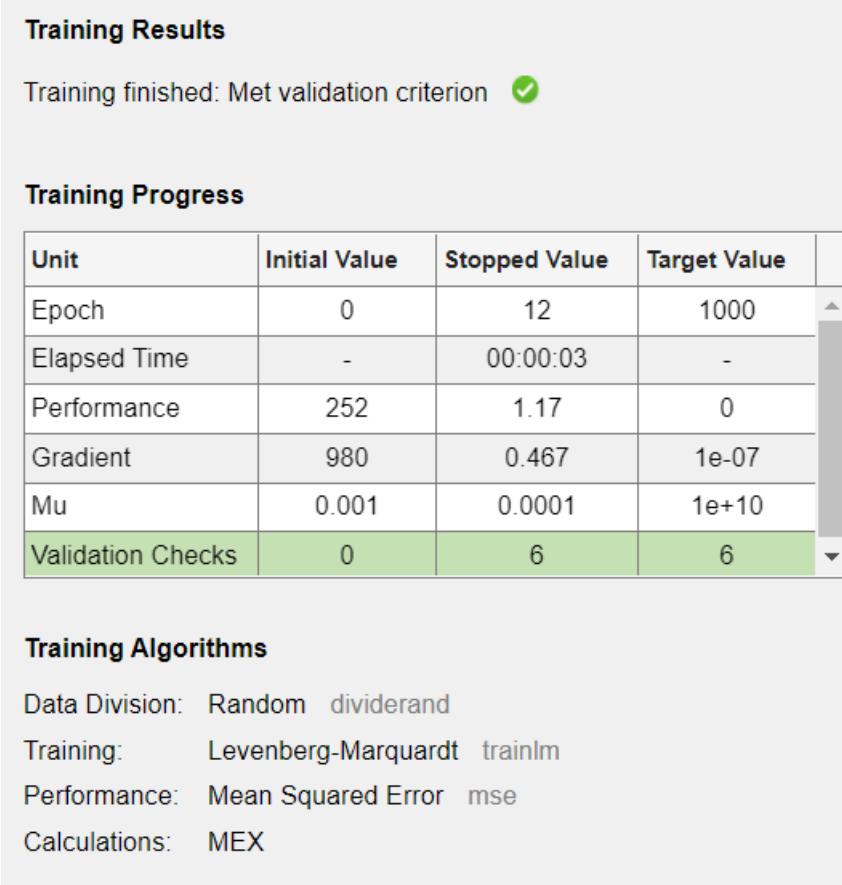


Figura 4.9: Resultados del entrenamiento de la red FeedForward para estimar la velocidad del rotor

Entre los resultados que nos muestra el entrenamiento podemos observar que el criterio de parada ha sido llegar a seis comprobaciones en la fase de validación en únicamente 12 épocas. El valor del gradiente se encuentra por debajo de 0.5 e igualmente el valor del rendimiento de la red está ligeramente por encima de 1 por lo que podemos deducir que el entrenamiento ha sido bueno y que la función para medir el rendimiento es el Error Cuadrático Medio.

Es importante recalcar que aunque el número de épocas sea bajo, ha sido suficiente para el entrenamiento de la red como podremos ver a continuación.

El entrenamiento también nos mostró cómo evolucionó el rendimiento de la red según iban avanzando las épocas, un histograma de errores y una regresión entre los valores estimados por la red NARX y los valores simulados por el motor.

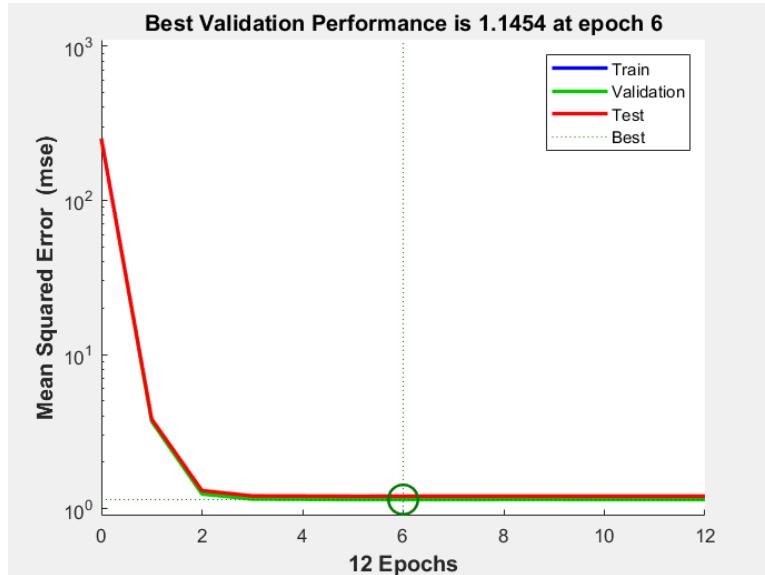
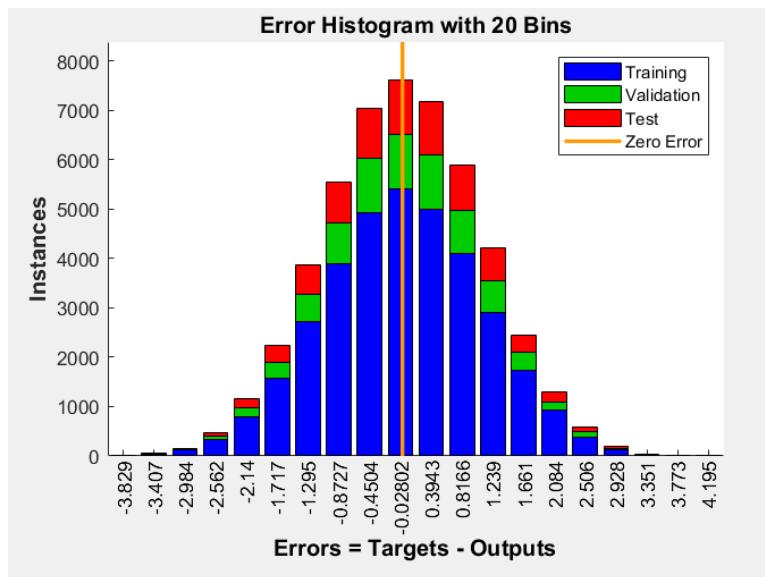
(a) Rendimiento NARX sobre ω_r (b) Histograma de Errores de NARX sobre ω_r

Figura 4.10: (a) Rendimiento de la red neuronal NARX para ω_r considerando ECM.
 (b) Histograma de Errores.

En la figura del rendimiento, se aprecia una importante mejora en el rendimiento en las primeras dos épocas ya que el valor baja de tener un orden de 10^2 a un valor cercano a 1 mientras que a partir de la tercera época el rendimiento se mantiene constante. A partir de la sexta época hasta la

duodécima época, el rendimiento se mantiene estable y, por tanto, se deja de entrenar la red.

En el histograma de errores podemos ver cómo la mayoría de los valores predichos tanto en la fase de entrenamiento, test y validación se diferencian de los valores reales menos de 2.14 unidades de diferencias y la diferencia más repetida es de 0.02802 unidades. Podemos decir que los valores predichos por la red neuronal son bastante similares a los valores simulados por el motor.

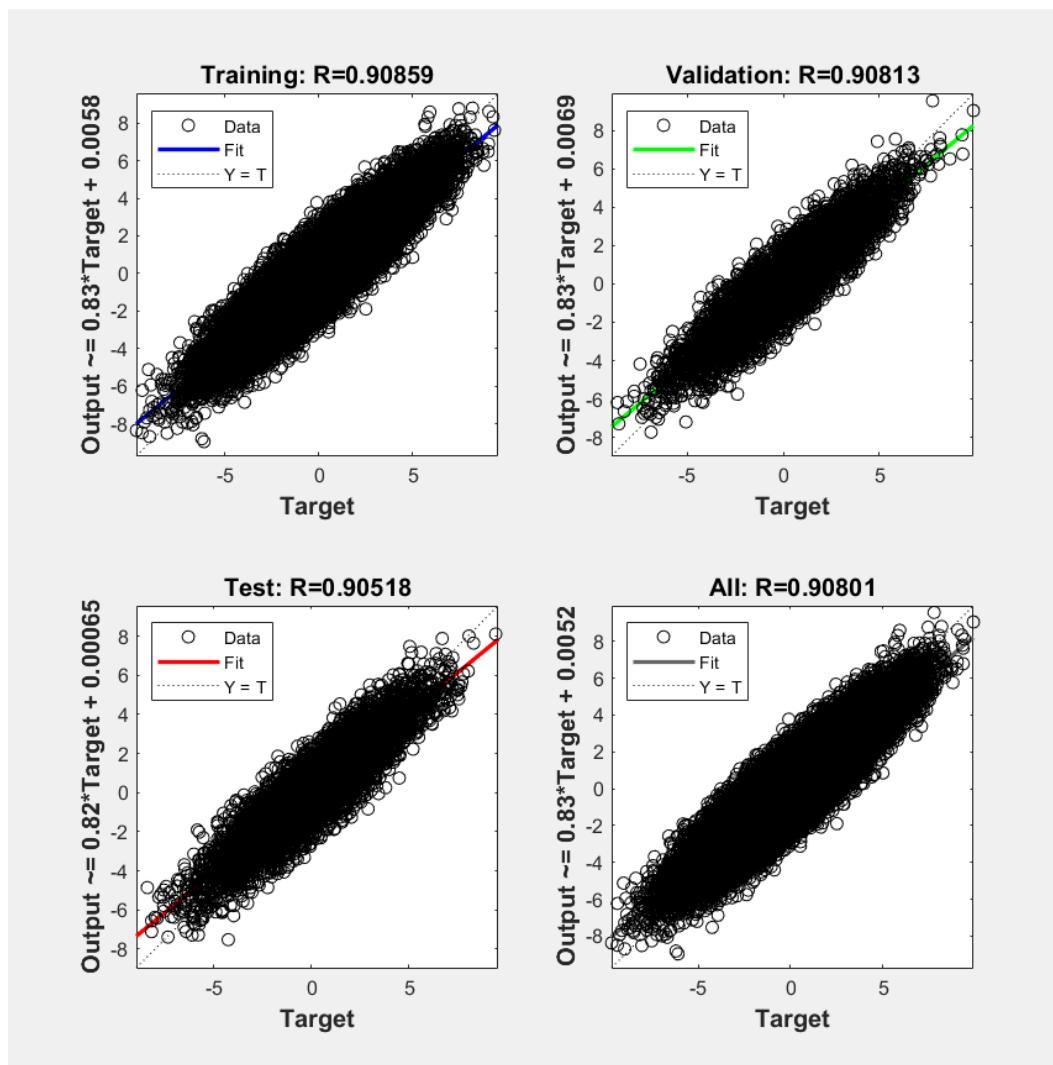


Figura 4.11: Regresión entre los valores estimados por la NARX y los valores simulados por el motor en la fase de entrenamiento, test, validación y en total.

Se observa que los puntos, aunque haya una gran cantidad y no pueda contemplarse con mucha claridad, siguen una recta. En esta figura, podemos observar cuatro regresiones: en el diagrama de arriba a la izquierda ha realizado la regresión con los datos de entrenamiento, en el diagrama de arriba a la derecha ha realizado la regresión sobre los datos que se han usado para la fase de validación, en el diagrama de abajo a la izquierda ha hecho la regresión con los datos de test y en el diagrama de abajo a la derecha hace la regresión con todos los datos que se han usado, es decir, con los datos de entrenamiento, validación y test juntos. Se observa que, a pesar de la gran cantidad de puntos que hay en cada gráfica, se sigue una recta en las cuatro gráficas. También podemos ver los valores de los coeficientes de correlación para la fase de entrenamiento, test, validación y en total son mayores que 0.9 por lo que existe cierta dependencia lineal entre las variables.

4.1.1.3. Resultados

Se ha usado MATLAB junto el paquete de herramientas DLT para validar todo el sistema de control asociado a las redes neuronales que han estimado la velocidad del rotor.

Las respuestas ofrecidas por las redes FeedForward y NARX que han estimado la velocidad del rotor han sido validadas considerando un tiempo de simulación de 250 segundos y un par de carga no constante que ha tomado valores entre $[-50, 50]$ de manera aleatoria durante el tiempo de simulación.

La velocidad del rotor junto con las perturbaciones del par de carga garantizan el comportamiento de las redes bajo muchas condiciones de operación diferentes.

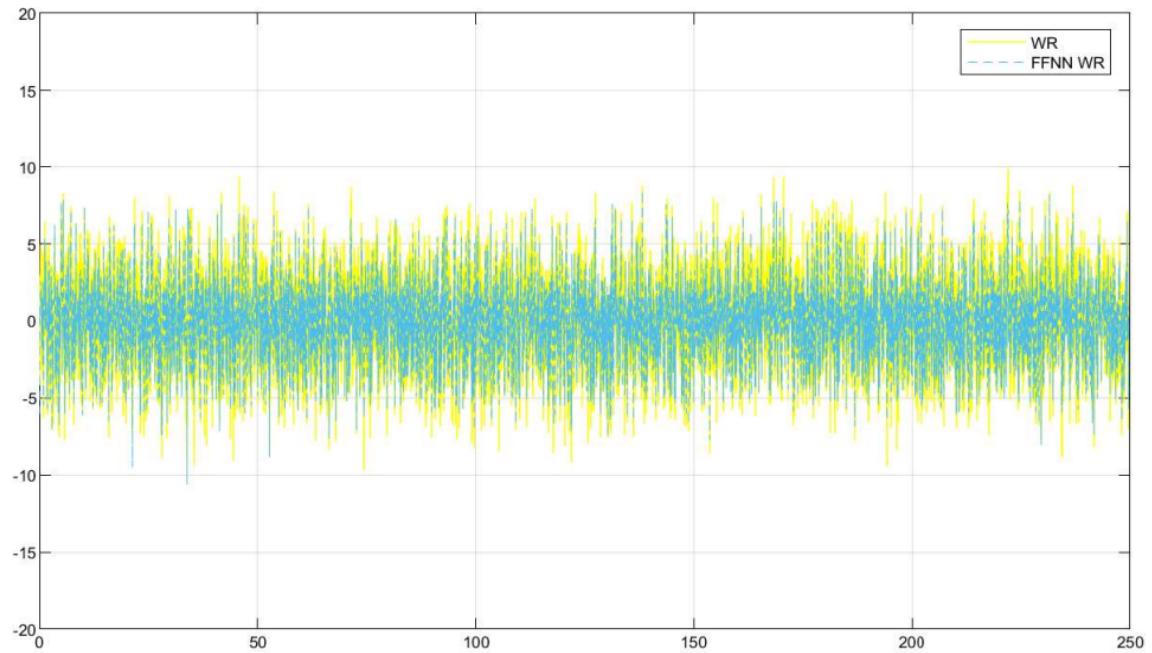


Figura 4.12: Velocidad del rotor, ω_r , (en amarillo) medida usando el modelo del motor en Simulink y (en azul) estimada por la FFNN.

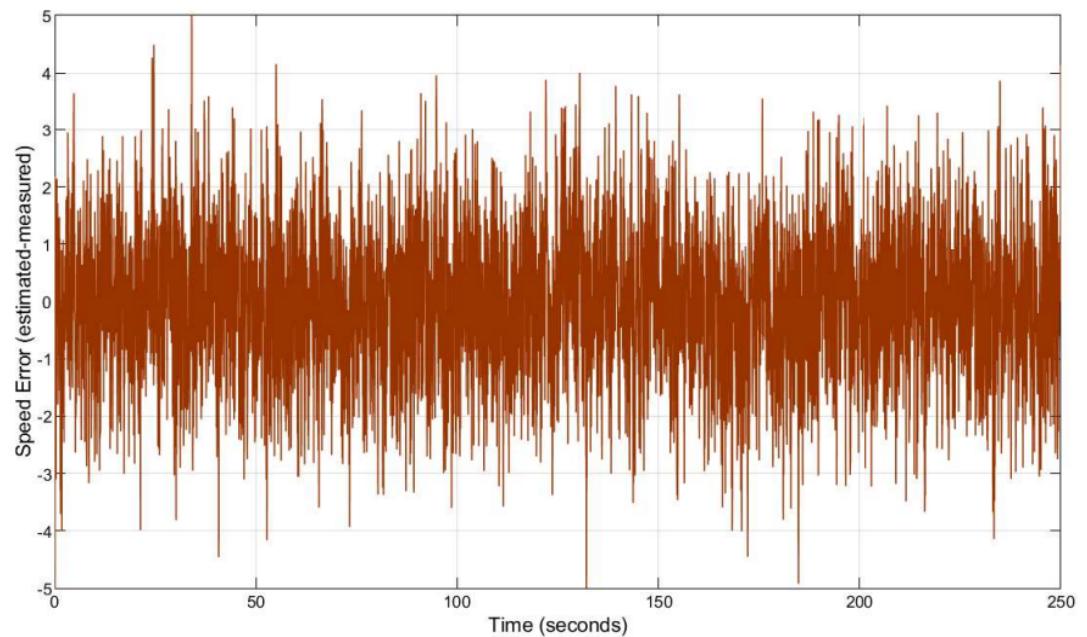


Figura 4.13: Error entre la velocidad del rotor medida y estimada por FFNN.

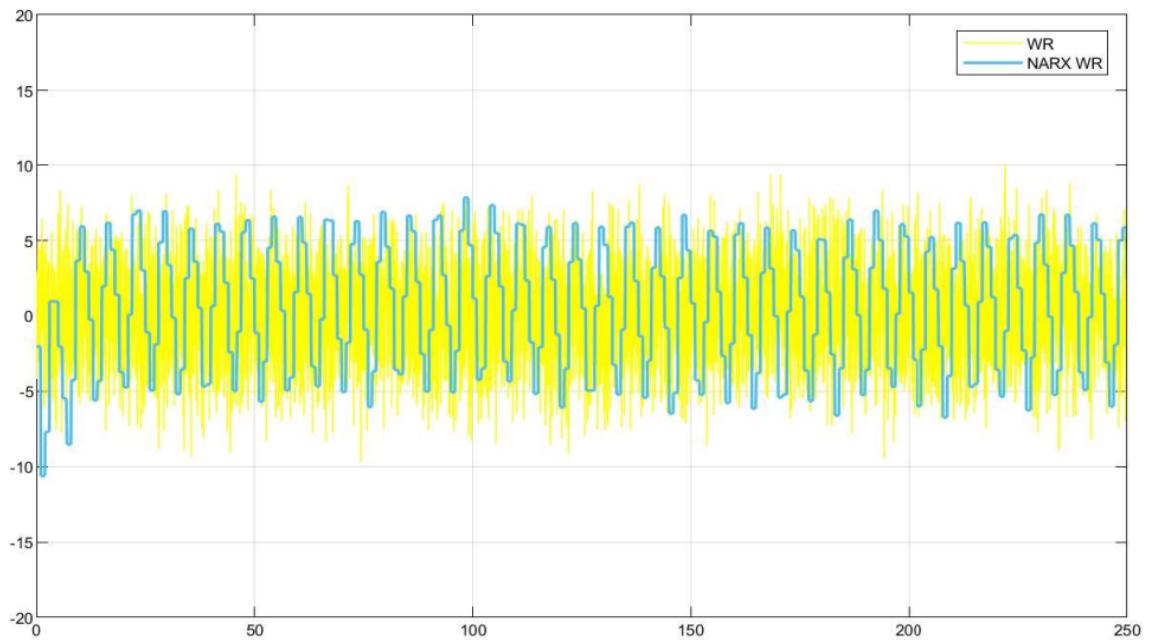


Figura 4.14: Velocidad del rotor, ω_r , (en amarillo) medida usando el modelo del motor en Simulink y (en azul) estimada por la NARX.

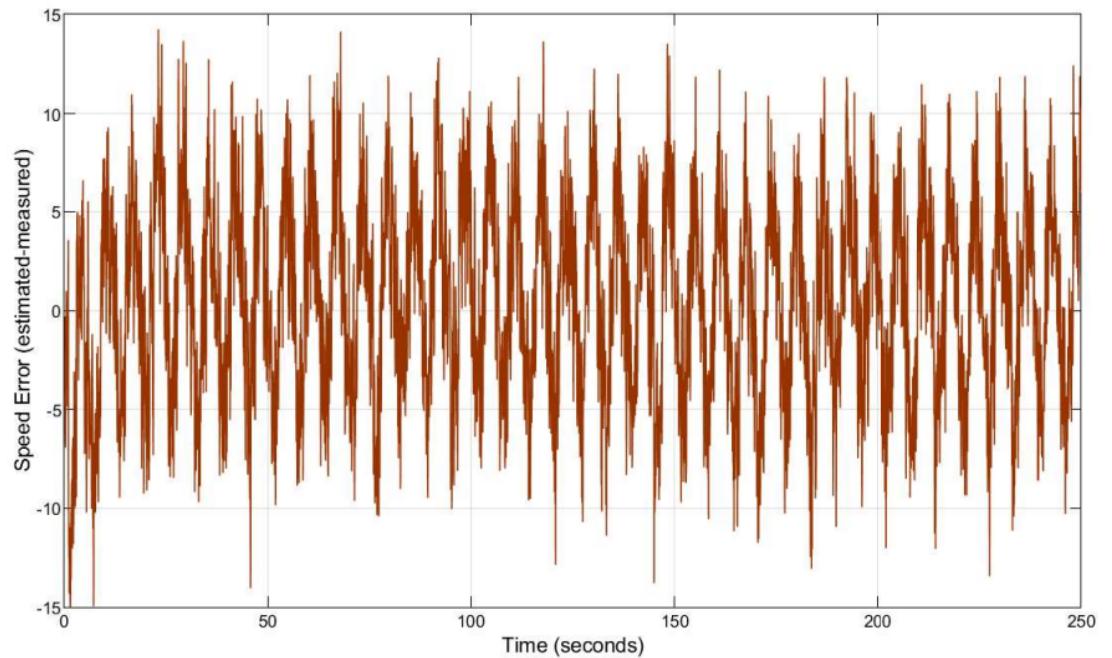


Figura 4.15: Error entre la velocidad del rotor medida y estimada por NARX.

La velocidad del rotor obtenida por el modelo y la estimada por la red FeedForward siguen la misma trayectoria y se mueven entre los mismos intervalos por lo que se puede validar la precisión de la red FeedForward como estimador de la velocidad del rotor.

Por otra parte, el rendimiento de la red NARX es bastante inferior a la red FeedForward. Sin embargo, para la red NARX también se ve que los valores estimados se mueven en el mismo intervalo que los valores medidos y siguen también la misma trayectoria pero con menos precisión que la red FeedForward. Aun así, podemos validar la precisión de la red NARX como estimador de la velocidad del rotor.

En 4.14 se aprecia durante los primeros 10 segundos de la simulación que los valores de la NARX son bastante inferiores a la velocidad del rotor real y como durante el resto de la simulación se mantiene sus valores en el intervalo donde se mueve los valores reales. A diferencia de lo mostrado en 4.12 en el que apreciamos un comportamiento bastante similar entre estimación y medición, en 4.14 no observamos esa precisión. Esto se puede deber al carácter recurrente de la NARX.

La precisión de ambas redes se puede contemplar más claramente en las figuras que miden el error. Para la red FeedForward, la mayoría de los errores toma valores en el intervalo $[-2, 2]$ mientras que para la red NARX la mayoría de los errores se encuentran en el intervalo $[-8, 10]$.

Otro factor importante a considerar es la robustez de ambas redes neuronales ya que no ocurre ninguna perturbación en los datos estimados que suponga valores anómalos respecto a la velocidad del motor.

Para representar en las gráficas el error cometido entre los valores estimados por las redes y los valores que obtenemos simulando el modelo se ha escogido el error absoluto. El error absoluto consiste en la diferencia en valor absoluto entre los valores reales menos los valores estimados.

$$Err_{Abs} = |Valor_{Real} - Valor_{Estim}| \quad (4.2)$$

Se ha usado el error absoluto y no el error relativo,

$$Err_{Rel} = \frac{|Valor_{Real} - Valor_{Estim}|}{Valor_{Real}} \quad (4.3)$$

porque aunque el error relativo no considera la escala, es muy sensible a valores cercanos a cero.

4.1.2. Redes neuronales para estimar el flujo magnético del rotor

Los valores de entrada que se han usado para entrenar las redes neuronales (FFNN y LRN) que van a estimar los flujos magnéticos del rotor han sido: el voltaje (v_α, v_β) medido en el marco de referencia $\alpha\beta$, la corriente (i_α, i_β), también medida en el marco $\alpha\beta$, y la velocidad del rotor (ω_r) y como datos de salida obtenemos los flujos magnéticos del rotor ($\lambda_{r\alpha}, \lambda_{r\beta}$) medido en el marco $\alpha\beta$.

Para estimar los valores de los flujos se ha usado como dato de entrada la velocidad del rotor que fue estimada por la red FeedForward y no el valor medido por el modelo.

4.1.2.1. FFNN para el flujo magnético del rotor

Para el diseño, entrenamiento, test y validación de la red FeedForward se ha usado la aplicación MATLAB junto al conjunto de herramientas DLT.

Los siguientes pasos muestran cómo se ha creado y diseñado la red FeedForward para estimar la velocidad del rotor:

Los pasos que se han seguido para crear y diseñar la red FeedForward para estimar los flujos magnéticos del rotor son los mismos pasos que se realizaron para la creación y diseño de la red FeedForward que estima la velocidad del rotor. Los únicos cambios que se han producido son los conjuntos de datos de entrada y salida. La configuración y entrenamiento de la red sería el siguiente:

- `ff_net = feedforwardnet([20,10]);`
- `ff_net.trainFcn='trainlm';`
- `ff_net.divideFcn='dividetrain';`
- `ff_net.divideMode='sample';`
- `ff_net.trainParam.Epochs=500;`
- `ff_net=configure(ff_net,transpose(InputDataFluxR),transpose(FluxR.Data));`
- `[ff_net,tr]=train(ff_net,transpose(InputDataFluxR),transpose(FluxR.Data));`
- `gensim(ff_net);`

De la misma forma que con la anterior red FeedForward, se ha creado una red neuronal FeedForward con dos capas ocultas y que contienen 20

neuronas y 10 neuronas respectivamente, se usa como función de entrenamiento el algoritmo de Levenberg-Marquardt, se realiza como máximo 500 épocas, se configura y entrena con el conjunto de datos de entrenamiento `InputDataFluxR` que contiene los voltajes y las tensiones en el marco $\alpha\beta$ y la velocidad del rotor estimada por la red `FeedForward` y como datos a aprender los flujos magnéticos del rotor en el marco $\alpha\beta$ que se encuentran almacenados en `FluxR.Data`. Con el comando `gensim(ff_net)` se convierte la red neuronal en un bloque de Simulink.

La estructura de la red neuronal se puede ver en la siguiente figura:

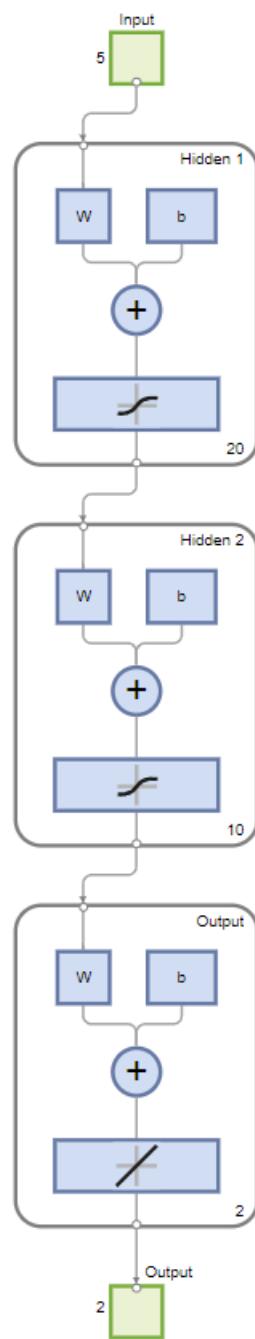


Figura 4.16: Estructura de la red neuronal FeedForward formada por 2 capas ocultas con 20 y 10 neuronas para estimar los flujos magnéticos del rotor.

La información que se ha obtenido tras el entrenamiento de la red fue el siguiente:

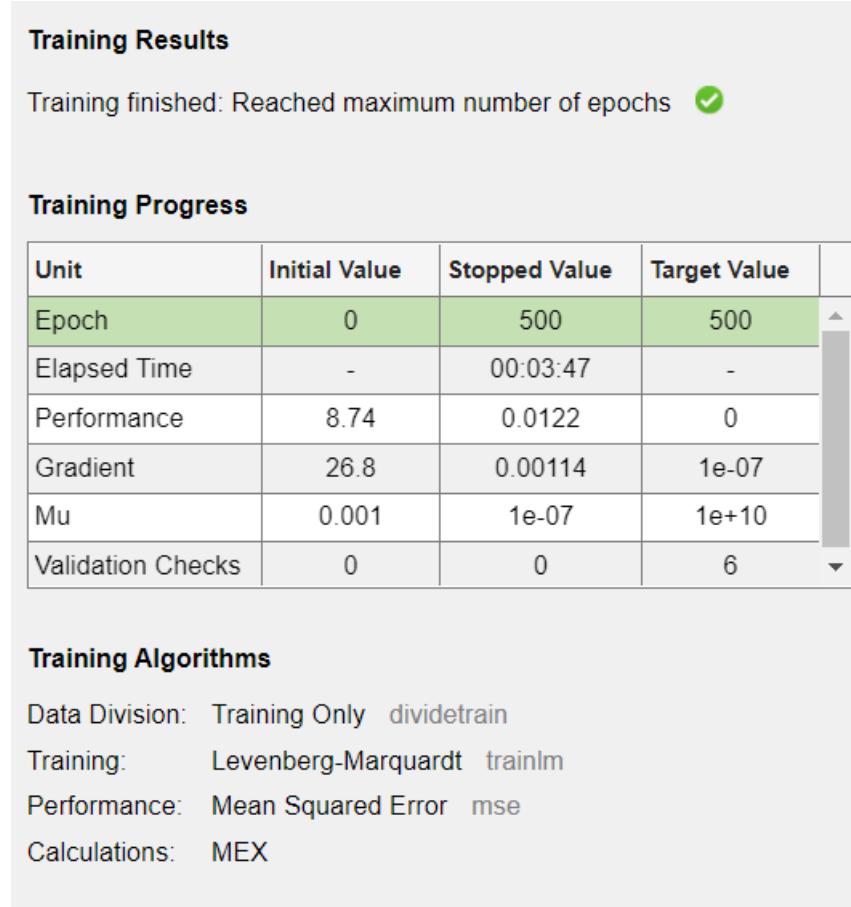


Figura 4.17: Resultados del entrenamiento de la red FeedForward para estimar los flujos magnéticos del rotor

Entre los resultados que nos muestra el entrenamiento podemos observar que el criterio de parada ha sido haber realizado el número máximo de épocas, el valor del gradiente se encuentra en el orden de 10^{-3} y el valor final del rendimiento es del orden de 10^{-2} por lo que podemos intuir que el entrenamiento ha sido bueno y que la función para medir el rendimiento es el Error Cuadrático Medio.

Asociado al entrenamiento, también obtenemos el rendimiento del entrenamiento de la red según las épocas, un histograma de errores con los errores más repetidos entre los datos reales y la estimación de la red y un gráfico de regresión entre los datos estimados y los datos reales.

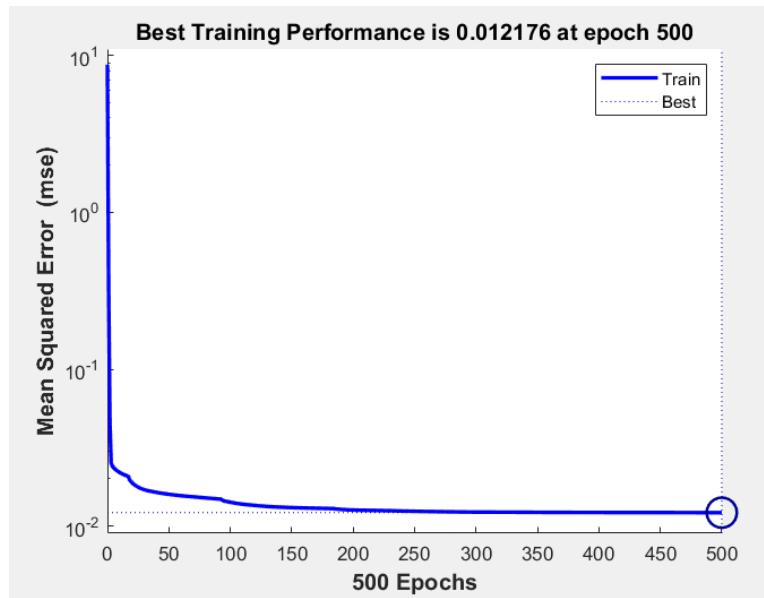


Figura 4.18: Rendimiento de la red FFNN para $\lambda_{r\alpha}, \lambda_{r\beta}$ considerando ECM

Se aprecia una importante mejora en el rendimiento en las primeras cinco épocas ya que el valor baja de tener un orden de 10^1 a un valor con un orden cercano a 10^{-2} , luego sigue decreciendo el rendimiento de manera mucho más moderada hasta la época 150, y desde esta época hasta la última época sigue decreciendo pero este descenso es casi imperceptible.

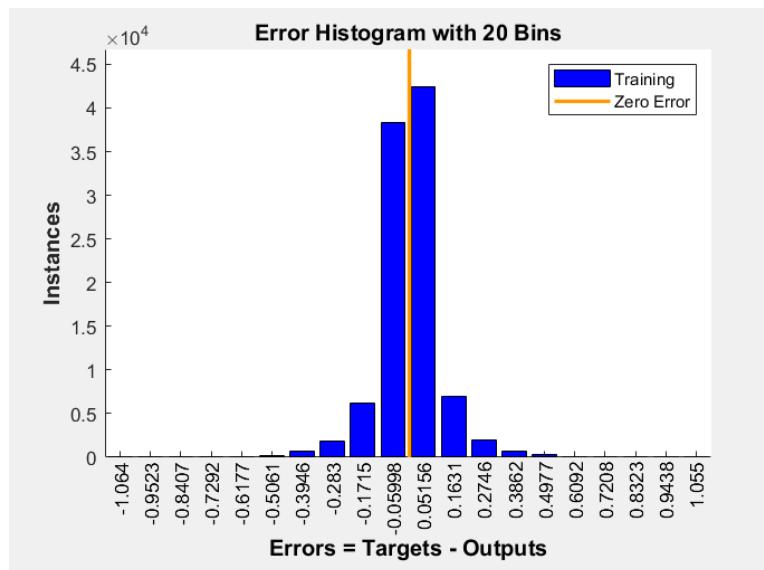


Figura 4.19: Histograma de Errores de FFNN sobre $\lambda_{r\alpha}, \lambda_{r\beta}$

En el histograma de errores podemos ver cómo la mayoría de los valores predichos se diferencian de los valores reales menos de 0.5061 unidades de diferencias y la diferencia más repetida es de 0.05156 unidades. Podemos decir que los valores predichos por la red neuronal son bastante similares a los valores simulados por el motor.

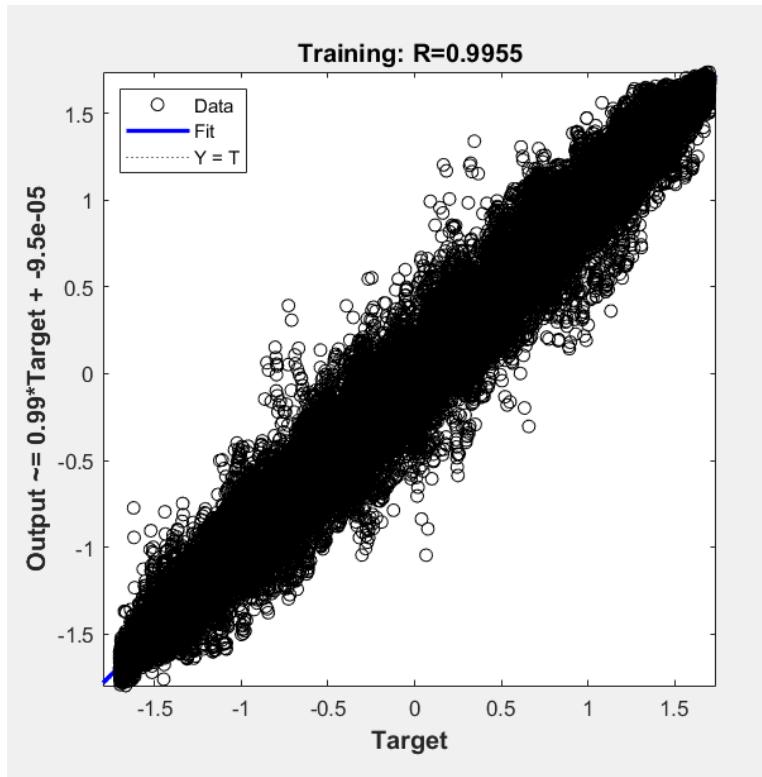


Figura 4.20: Regresión datos FFNN y datos $\lambda_{r\alpha}, \lambda_{r\beta}$

En la figura de la regresión, aunque haya una gran cantidad de puntos que aparecen en la gráfica, observamos que siguen una recta. Se nos aporta el valor del coeficiente de correlación que es $R = 0.9955$, prácticamente el valor de R es 1, por lo que podemos afirmar que existe una dependencia lineal entre los flujos magnéticos del rotor estimados por la red FFNN y los valores del flujo magnético del rotor simulados por el modelo.

4.1.2.2. LRN para el flujo magnético del rotor

Para la estimación del flujo magnético del rotor con una red recurrente se usó, previamente, una red NARX al igual que con la estimación para la velocidad del rotor. Debido a los pobres resultados en la estimación del flujo

magnético del rotor mostrados por la red NARX, indagué sobre otras redes neuronales recurrentes hasta encontrar las redes LRN. El resultado de la estimación por la red LRN superó al resultado de la red NARX y por este motivo se ha usado una red LRN como red recurrente para la estimación de los flujos magnéticos del rotor.

Al igual que para la red FeedForwaard, el diseño, entrenamiento, test y validación de la red LRN se ha usado la aplicación MATLAB junto al conjunto de herramientas DLT.

Los siguientes pasos muestran cómo se ha creado y diseñado la red LRN para estimar la velocidad del rotor:

- delay=[1:2];
- lrn_net = narxnet(delay,11);
- lrn_net.trainFcn='trainbr';
- lrn_net.trainParam.epochs=50;
- lrn_net = train(lrn_net,F,S)
- gensim(lrn_net);

De esta forma, se crea una red neuronal LRN con una capa oculta que contienen 11 neuronas y se le aplica un retardo [1:2] a los datos que se obtienen de la primera capa, se usa como función de entrenamiento el algoritmo de Regularización Bayesiana y se indica que se realice como máximo 50 épocas. Para el entrenamiento de la red se requiere el nombre de la red y los objetos F y S. F se corresponde con una celda de forma de array que contiene los datos de entrada ($v_\alpha, v_\beta, i_\alpha, i_\beta, \omega_r$) y S, también es una celda de forma de array y contiene los datos de salida ($\lambda_{r\alpha}, \lambda_{r\beta}$). Finalmente, con `gensim(lrn_net)` se genera el bloque Simulink de la red neuronal.

Debido a que en las redes LRN la retroalimentación se realiza en cada capa, a diferencia de la red NARX que únicamente toma el valor final y lo utiliza como dato de entrada, se han reducido el número de épocas para el entrenamiento de la red.

La estructura de la red neuronal se puede ver en la siguiente figura:

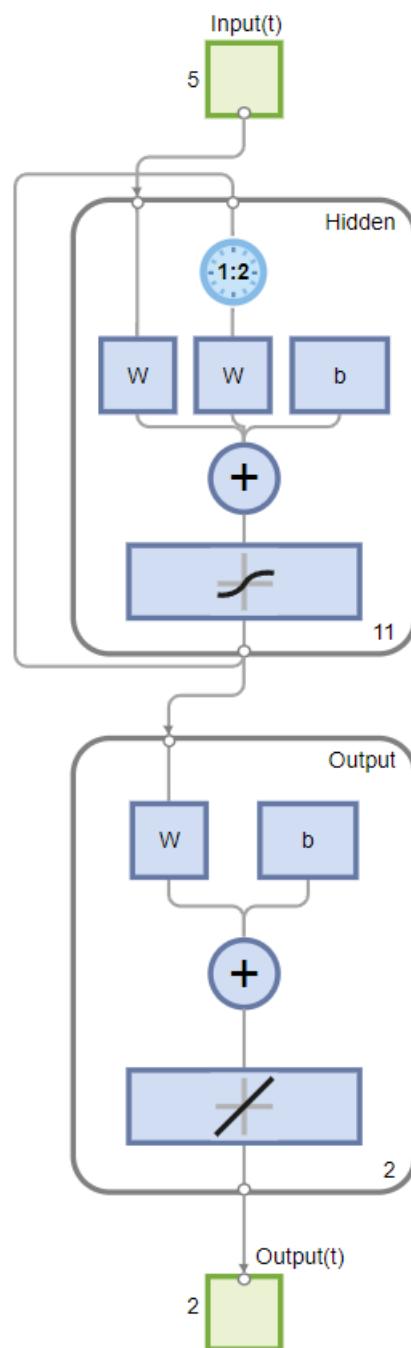


Figura 4.21: Estructura de la red neuronal LRN formada por una capa oculta con 11 neuronas para estimar los flujos magnéticos del rotor.

La información que se ha obtenido tras el entrenamiento de la red fue el

siguiente:

Training Progress			
Unit	Initial Value	Stopped Value	Target Value
Epoch	0	50	50
Elapsed Time	-	00:02:29	-
Performance	8.02	0.00185	0
Gradient	20.2	0.338	1e-07
Mu	0.005	5e+04	1e+10
Effective # Param	332	281	0
Sum Squared P...	43.8	20.9	0

Training Algorithms			
Data Division:	Random	dividerand	
Training:	Bayesian Regularization	trainbr	
Performance:	Mean Squared Error	mse	
Calculations:	MEX		

Figura 4.22: Resultados del entrenamiento de la red LRN para estimar los flujos magnéticos del rotor

Entre los resultados que nos muestra el entrenamiento podemos observar que el criterio de parada ha sido el haber realizado el número máximo de épocas, el valor del gradiente se encuentra en el orden de 10^{-1} , igualmente el valor del rendimiento de la red es un número del orden de 10^{-3} (0.00185) por lo que podemos deducir que el entrenamiento ha sido bueno y también observamos como la suma de los cuadrados (última fila de la tabla) ha alcanzado un valor menor al que tenía en el inicio.

El entrenamiento también nos mostró cómo evolucionó el rendimiento de la red según iban avanzando las épocas, un histograma de errores y una regresión entre los valores estimados por la red LRN y los valores simulados por el motor.

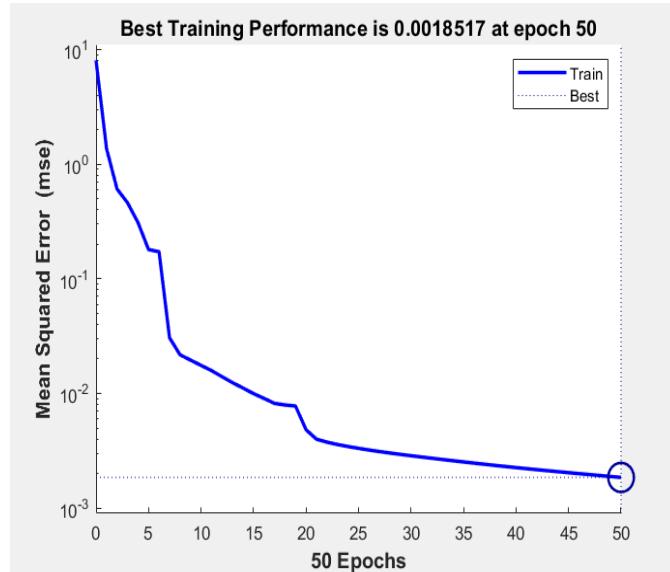
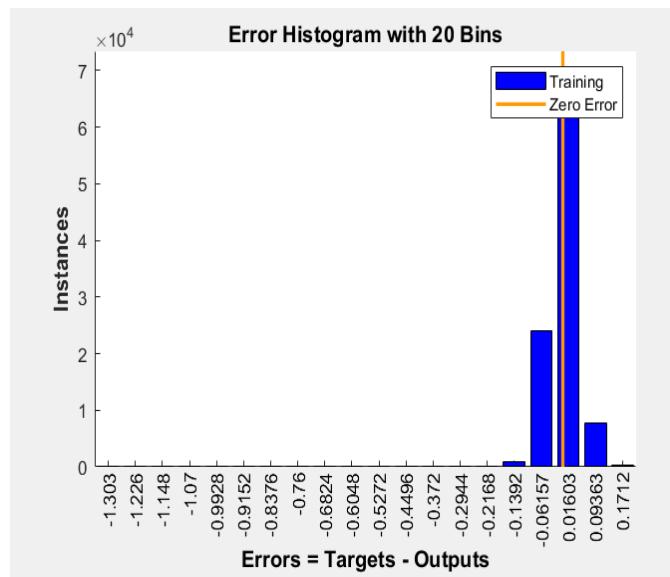
(a) Rendimiento LRN sobre $\lambda_{r\alpha}$ y $\lambda_{r\beta}$ (b) Histograma de Errores de NARX sobre $\lambda_{r\alpha}$ y $\lambda_{r\beta}$

Figura 4.23: (a) Rendimiento de la red neuronal LRN para $\lambda_{r\alpha}$ y $\lambda_{r\beta}$ considerando ECM. (b) Histograma de Errores.

En la figura del rendimiento, se aprecia una importante mejora en el rendimiento en las primeras veinte épocas observando que pasa de un valor de orden 10^1 a un valor de orden 10^{-2} . A partir de esta vigésima época hasta la quincuagésima época, notamos un ligero descenso del error pero no tan

pronunciado como en las primeras veinte épocas.

En el histograma de errores podemos ver cómo la mayoría de los valores predichos se diferencian de los valores reales menos de 0.1712 unidades de diferencias y la diferencia más repetida es de 0.01603 unidades. Podemos afirmar que los valores predichos por la red neuronal son bastante similares a los valores simulados por el motor.

Un detalle curioso del histograma es que la gran mayoría de los errores que se han producido han sido por sobrepasar el valor verdadero. Esto se puede apreciar bien en 4.31, en cada subida del valor del flujo magnético, la estimación de la LRN siempre muestra un valor superior al real. Así que al hacer, $\text{Errores} = \text{Objetivos} - \text{Salidas}$ la mayoría de los valores van a ser negativos ya que la estimación es mayor que el valor real. Por esta razón, el histograma no está centrado y las barras con los errores más frecuentes están en el lado derecho.

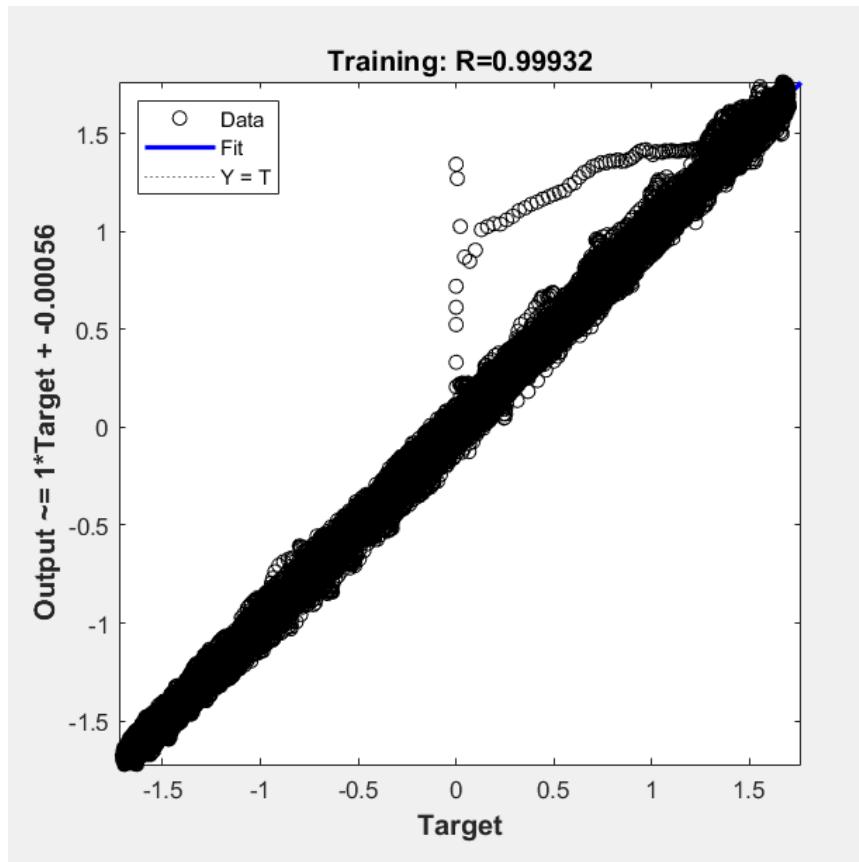


Figura 4.24: Regresión entre los valores estimados por la LRN y los valores simulados por el motor.

Se observa, de manera muy clara, que los puntos siguen una recta. Aun así, vemos que hay algunos valores anómalos pero son muy pocos por lo que no afecta mucho en el estudio de la regresión. Estos valores anómalos se podrían eliminar ya que, si observamos las figuras 4.29 y 4.31, no apreciamos que estos valores anómalos se correspondan con valores simulados por el modelo. No se han eliminado estos valores debido a que el proceso para eliminarlos sería bastante tedioso. Además vemos que el valor del coeficientes de regresión, $R = 0.99932$, es prácticamente 1 por lo que podemos decir que existe una relación de linealidad entre los valores estimados y los reales.

4.1.2.3. Resultados

Se ha usado MATLAB junto el paquete de herramientas DLT para validar todo el sistema de control asociado a las redes neuronales que han estimado los flujos magnéticos del rotor.

Las respuestas ofrecidas por las redes FeedForward y LRN que han estimado los flujos magnéticos del rotor han sido validadas considerando un tiempo de simulación de 250 segundos y un par de carga no constante que ha tomado valores entre $[-50, 50]$ de manera aleatoria durante el tiempo de simulación.

La velocidad del rotor junto con las perturbaciones del par de carga garantizan el comportamiento de las redes bajo muchas condiciones de operación diferentes.

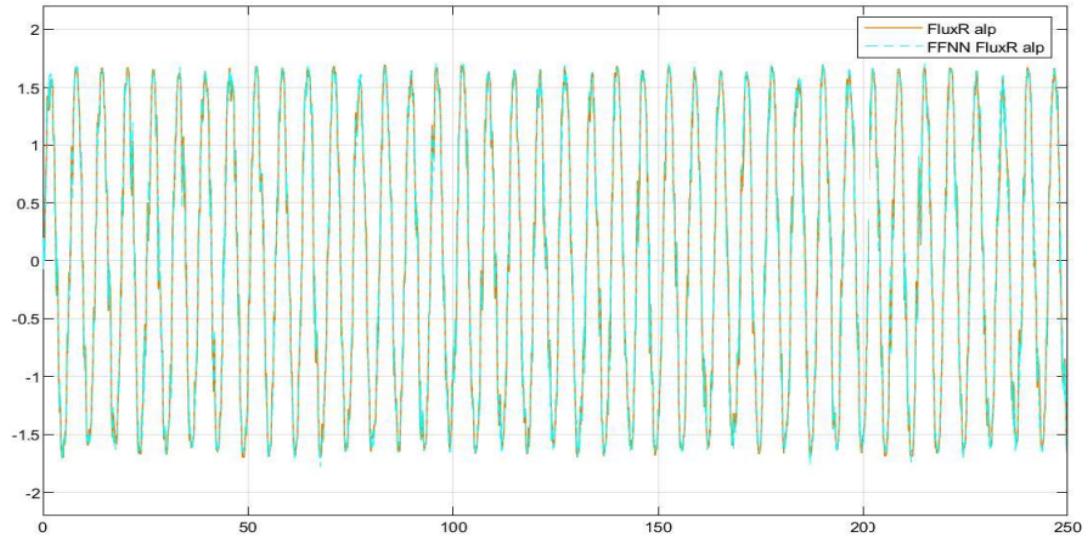


Figura 4.25: Flujo magnético del rotor en la componente α , $\lambda_{r\alpha}$, (en naranja) medida usando el modelo del motor en Simulink y (en azul) el estimado por la FFNN.

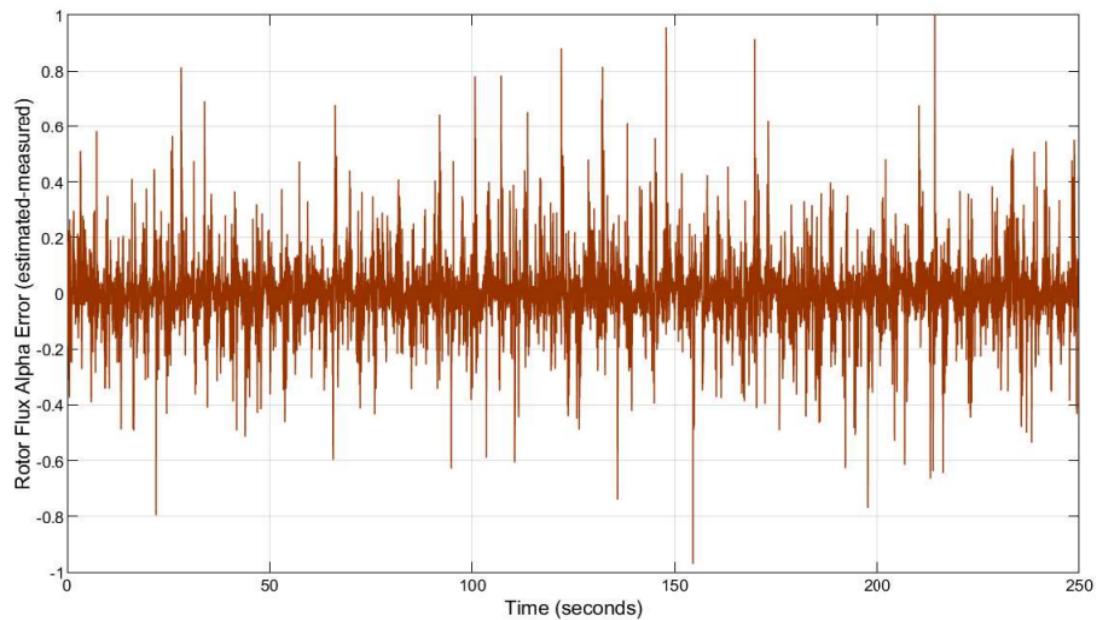


Figura 4.26: Error entre el flujo magnético del rotor medida en la componente α y el estimado por FFNN.

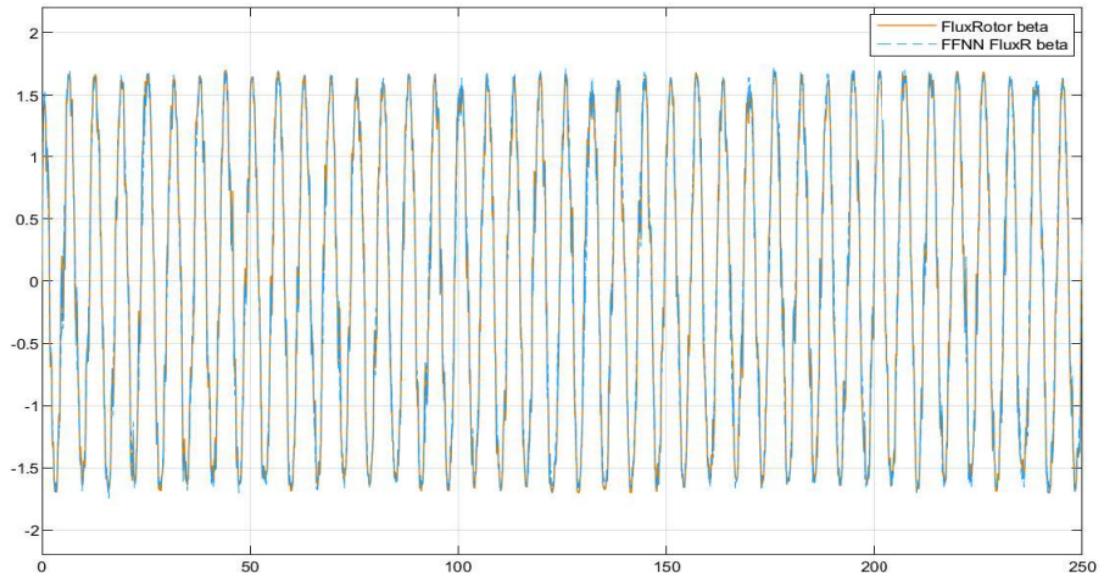


Figura 4.27: Flujo magnético del rotor en la componente β , $\lambda_{r\beta}$, (en naranja) medida usando el modelo del motor en Simulink y (en azul) el estimado por la FFNN.

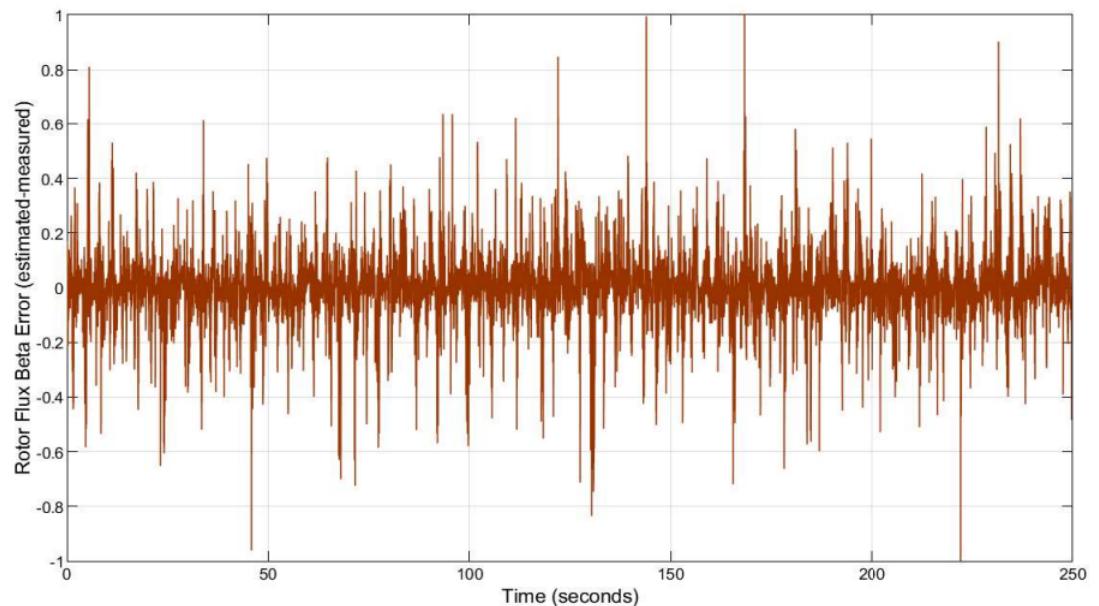


Figura 4.28: Error entre el flujo magnético del rotor medida en la componente β y el estimado por FFNN.

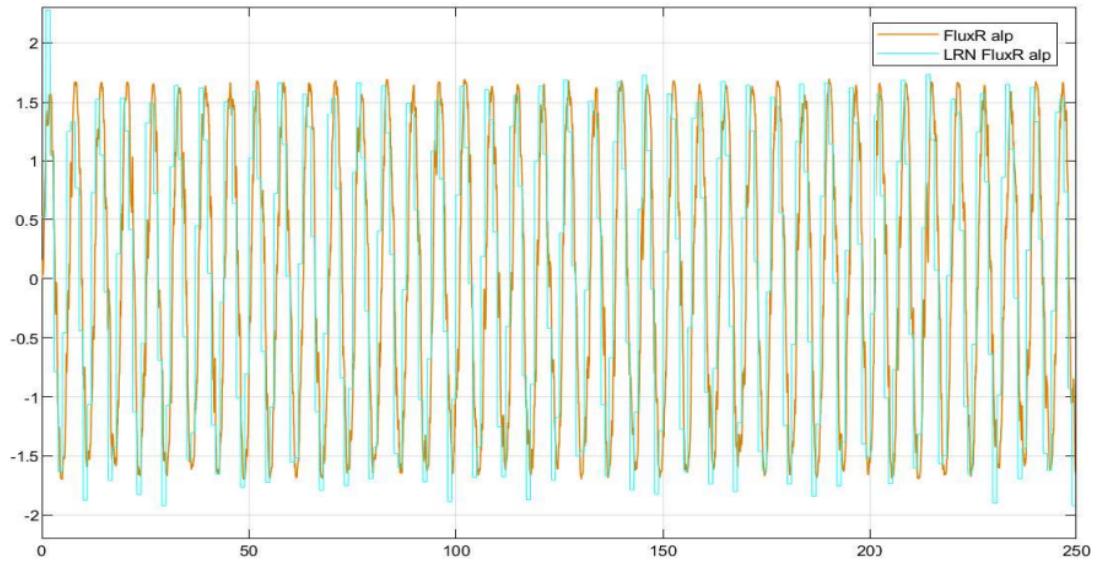


Figura 4.29: Flujo magnético del rotor en la componente α , $\lambda_{r\alpha}$, (en naranja) medida usando el modelo del motor en Simulink y (en azul) el estimado por la LRN.

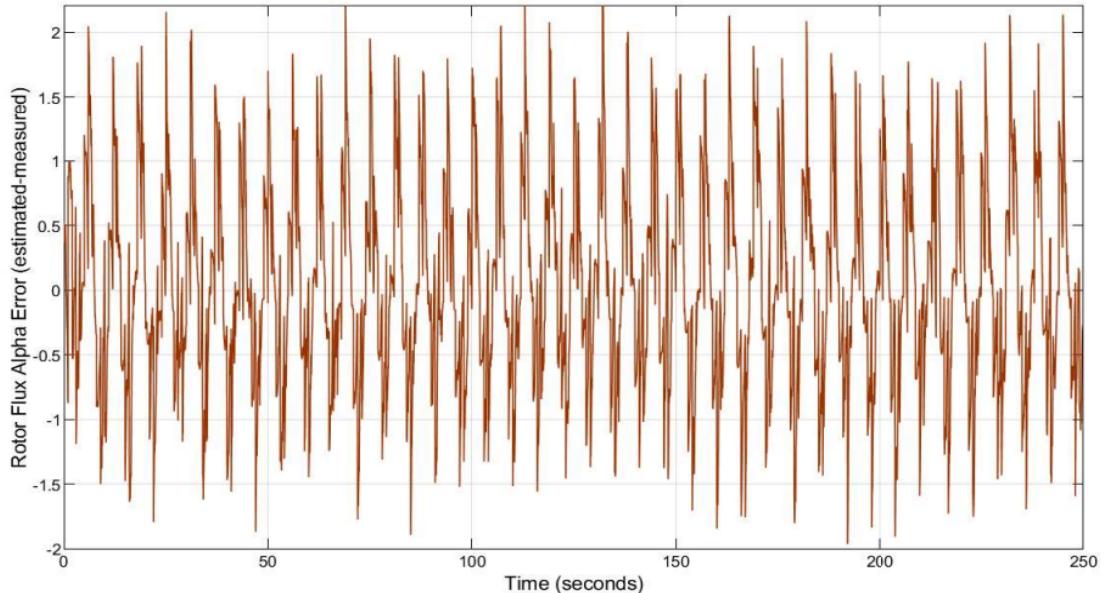


Figura 4.30: Error entre el flujo magnético del rotor medida en la componente α y el estimado por LRN.

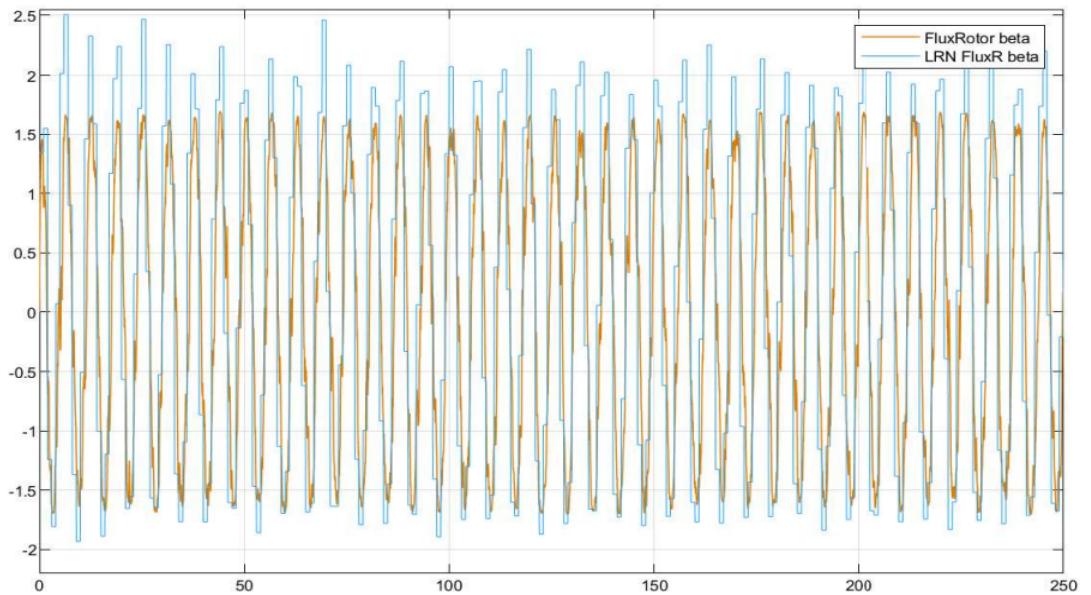


Figura 4.31: Flujo magnético del rotor en la componente β , $\lambda_{r\beta}$, (en naranja) medida usando el modelo del motor en Simulink y (en azul) el estimado por la LRN.

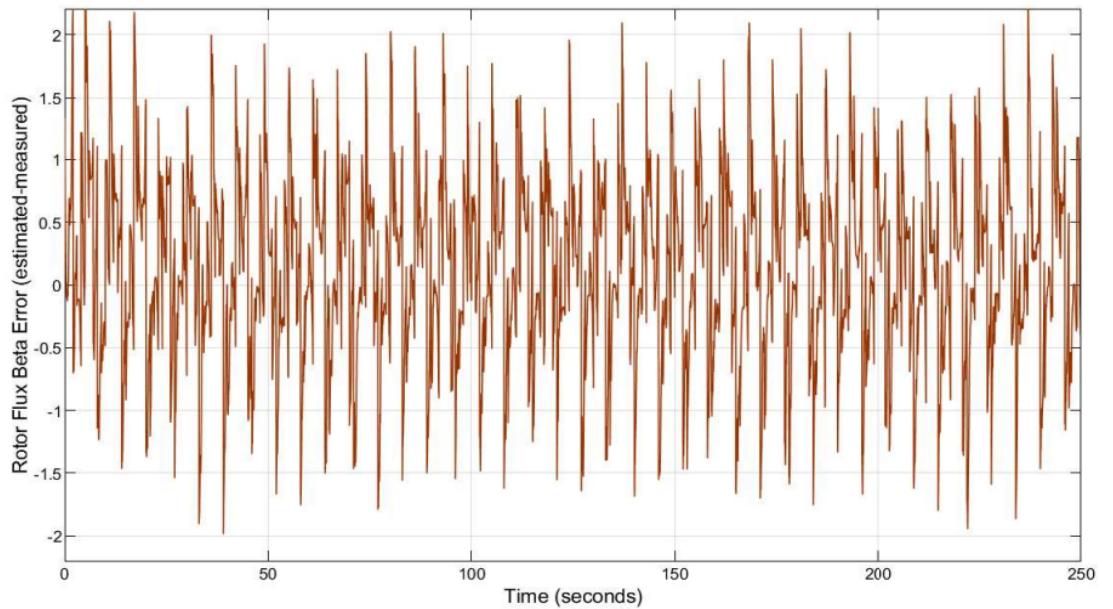


Figura 4.32: Error entre el flujo magnético del rotor medida en la componente β y el estimado por LRN.

Los flujos magnéticos del rotor obtenidos por el modelo y los estimados por la red FeedForward siguen la misma trayectoria y se mueven entre los mismos intervalos por lo que se puede validar la precisión de la red FeedForward como estimador de los flujos magnéticos del rotor. Esto se puede apreciar tanto en la figura 4.25 como en la figura 4.27.

Por otra parte, al observar las figuras 4.29 y 4.31, apreciamos que los valores estimados por la red LRN siguen la misma trayectoria que los flujos magnéticos medidos y toman sus valores prácticamente en los mismos intervalos aunque en 4.31 sobreestima los valores máximos. Aun así, podemos validar la precisión de la red LRN como estimador de los flujos magnéticos del rotor.

La precisión de ambas redes se puede contemplar más claramente en las figuras que miden el error, 4.26, 4.284.30, 4.32. Para la red FeedForward, la mayoría de los errores toma valores en el intervalo $[-0.6, 0.6]$ tanto para $\lambda_{r\alpha}$ como para $\lambda_{r\beta}$ mientras que para la red LRN la mayoría de los errores se encuentran en el intervalo $[-1.5, 1.5]$.

En 4.29 podemos ver como en los primeros segundos el valor estimado difiere bastante con el valor medido y en el resto de la simulación difiere con los valores mínimos en un rango menor, en 4.31 vemos que en los valores máximos, la red ha producido valores un 0.5 mayores que los medidos mientras que en 4.25 y 4.27 no notamos estas diferencias. Es decir, la estimación ha predicho mejor los valores del flujo magnético del rotor en la componente α que en la componente β . Esto se debe a que en los primeros segundos de la simulación, la subida del flujo magnético del rotor en la componente β es más abrupta que en la componente α lo que provoca un período distinto en la primera onda y conlleva una peor estimación.

Otro factor importante a considerar es la robustez de ambas redes neuronales ya que no ocurre ninguna perturbación en los datos estimados que suponga valores anómalos respecto a la velocidad del motor.

Al igual que en el caso de la velocidad del rotor y por los mismos motivos, para representar en las gráficas el error cometido entre los valores estimados por las redes y los valores que obtenemos simulando el modelo se ha escogido el error absoluto.

Tras los resultados expuestos podemos afirmar que se puede hacer uso de redes neuronales como estimadores de la velocidad del rotor y de los flujos magnéticos del rotor en un motor de inducción. Se ha comprobado que tanto en redes prealimentadas como en redes recurrentes pueden estimar estos parámetros de un motor de inducción. No obstante hay que destacar que el rendimiento de las redes prealimentadas ha sido superior al de las redes recurrentes. Es fácilmente comprobable observando los límites de las gráficas

de los errores 4.13, 4.15, 4.26, 4.28, 4.30 y 4.32. Mientras los errores en la red FeedForward para la velocidad del rotor se encuentran en el intervalo $[-2, 2]$ y para los flujos magnéticos del rotor en el intervalo $[-0.6, 0.6]$, para las redes recurrentes estos límites son mayores. Los errores en la red NARX para la estimación de la velocidad del rotor se mueven en el intervalo $[-8, 10]$ y los errores en la red LRN para la estimación de los flujos magnéticos del rotor se mueven en el intervalo $[-1.5, 1.5]$.

Bibliografía

- [1] Wisam Abed. «Speed Control of DC Motor Using Adaptive Neuro Fuzzy Controller WISAM NAJM AL-DIN ABED». En: *The Journal of Scientific and Engineering Research* (ene. de 2015).
- [2] A.A.Z. Diab - M.A. Elsawy- K.A. Denis - S. Alkhalfaf - Z.M. Ali. «Artificial Neural Based Speed and Flux Estimators for Induction Machine Drives with Matlab/Simulink.» En: *Mathematics 2022, 10, 1348.* (abr. de 2022). DOI: <https://doi.org/10.3390/math10081348>.
- [3] M.T. Hagan - H.B. Demuth - M. Beale. En: *Neural Network Design.* PWS Publishing Co, 1996.
- [4] V Prasad - BW Bequette. «Nonlinear System Identification and Model Reduction Using Artificial Neural Networks». En: *Computers & Chemical Engineering* (dic. de 2003).
- [5] F. Blaschke. «The principle of field orientation as applied to the new transvector closed –loop control system for rotating-field machines». En: *Siemens Rev Vol. 39.Nº. 3* (mayo de 1972), págs. 217-220.
- [6] Fernando Ceballos - Luís Eduardo Muñoz - Julián Moreno Cadavid. «Selección de perceptrones multicapa usando aprendizaje bayesiano». En: *Scientia et Technica 3.49* (dic. de 2011), págs. 110-116. DOI: 10.22517/23447214.1489. URL: <https://revistas.utp.edu.co/index.php/revistaciencia/article/view/1489>.
- [7] Eduardo Camacho y Carlos Bordons. *Model Predictive Control.* Vol. 13. Ene. de 2004. ISBN: 978-0-85729-398-5. DOI: 10.1007/978-0-85729-398-5.
- [8] M.I. Capel. «Artificial Neuron-Based Model for a Hybrid Real-Time System: Induction Motor Case Study». En: *Mathematics 2022, 10, 3410.* (sep. de 2022). DOI: <https://doi.org/10.3390/math10183410>.
- [9] M Lee - HS Choi. «A Robust Neural Controller for Underwater Robot Manipulators». En: *IEEE Transactions on Neural Networks* (nov. de 2000).

- [10] F. Dan Foresee y M.T. Hagan. «Gauss-Newton approximation to Bayesian learning». En: *Proceedings of International Conference on Neural Networks (ICNN'97)*. Vol. 3. 1997, págs. 1930-1935. DOI: 10.1109/ICNN.1997.614194.
- [11] Ahmed Gamal - Hegazy Rezk - Ahmed Diab. «Robust Sensorless Model-Predictive Torque Flux Control for High-Performance Induction Motor Drives». En: *Mathematics* Vol. 9 (feb. de 2021), pág. 403. DOI: <https://doi.org/10.3390/math9040403>.
- [12] Jeffrey L. Elman. «Finding Structure in Time». En: *Cognitive Science* 14.2 (1990), págs. 179-211. DOI: https://doi.org/10.1207/s15516709cog1402_1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog1402_1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.
- [13] Luca Franceschi et al. «Forward and Reverse Gradient-Based Hyperparameter Optimization». En: *Proceedings of the 34th International Conference on Machine Learning*. Ed. por Doina Precup y Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, jun. de 2017, págs. 1165-1173. URL: <https://proceedings.mlr.press/v70/franceschi17a.html>.
- [14] J Chen - TC Huang. «Applying Neural Networks to On-line Updated PID Controllers for Nonlinear Process Control». En: *Journal of Process Control* (2004).
- [15] Kangbeom Cheon - Jaehoon Kim - Moussa Hamadache - Dongik Lee. «On Replacing PID Controller with Deep Learning Controller for DC Motor System». En: *Journal of Automation and Control Engineering* Vol. 3.Nº. 6 (dic. de 2015). DOI: doi:10.12720/joace.3.6.452-456.
- [16] Kumpati S. Narendra y Kannan Parthasarathy. «Learning automata approach to hierarchical multiobjective analysis». En: *IEEE Trans. Syst. Man Cybern.* 21 (1991), págs. 263-272.
- [17] Özcan Otkun, Faruk Demir y Selçuk Otkun. «Scalar speed control of induction motor with curve-fitting method». En: *Automatika* 63 (dic. de 2022), págs. 618-626. DOI: 10.1080/00051144.2022.2060657.
- [18] Ahmed A. Zaki Diab - Vladimir V. Pankratov. «Model Predictive Control of Vector controlled Induction Motor Drive». En: *IEEE 11th International Conference on Actual Problems of Electronics Instrument Engineering (APEIE)* (2012), págs. 127-134. DOI: 10.1109/APEIE.2012.6629086.

- [19] Fabian Pedregosa. «Hyperparameter optimization with approximate gradient». En: *Proceedings of the 33nd International Conference on Machine Learning (ICML)*. 2016. URL: <http://proceedings.mlr.press/v48/pedregosa16.html>.
- [20] Adel Aktaibi - Daw Ghanim - M. A. Rahman. «Dynamic Simulation of a Three-Phase Induction Motor Using Matlab Simulink». En: nov. de 2011. DOI: 10.13140/RG.2.1.2705.4243.
- [21] Shai Shalev-Shwartz y Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014, págs. I-XVI, 1-397. ISBN: 978-1-10-705713-5.
- [22] A. Ba-Razzouk - A. Cheriti - G. Olivier - P. Sicard. «Field oriented control of induction motors using neural networks decouplers, Industrial Electronics, Control, and Instrumentation, 1995.» En: *Proceedings of the 1995 IEEE IECON 21st International Conference*. Vol. 2 (nov. de 1995), págs. 1428-1433.
- [23] Pan Zhao et al. «Design of a Control System for an Autonomous Vehicle Based on Adaptive-PID». En: *International Journal of Advanced Robotic Systems* 9 (jul. de 2012), pág. 1. DOI: 10.5772/51314.

Apéndice: Parámetros y valores del motor de inducción

Los parámetros del motor de inducción son los siguientes:

- R_r , resistencia del rotor (Ω)
- R_s , resistencia del estator (Ω)
- L_r , inductancia del rotor (H)
- L_s , inductancia del estator (H)
- L_m , inductancia de magnetización (H)
- J , momento de inercia ($kg \cdot m^2$)
- f_b , frecuencia básica de las fases del motor de inducción (Hz)
- T_l , par de carga (Nm)
- T_e , par electromagnético (Nm)
- T_t , tiempo constante del rotor ($T_r = \frac{L_r}{R_r}$)
- δ , coeficiente de pérdidas ($\delta = 1 - \frac{L_m^2}{L_s L_r}$)
- f_d , coeficiente de pérdida de la carga (Ns/m)
- n , número de polos
- ω_r , velocidad del rotor (rad/s)
- ω_e , velocidad de sincronismo ($\omega_e = \frac{120 \cdot 2\pi \cdot f_b}{60 \cdot n}$) (rad/s)
- k_t , constante de magnetización ($k_t = \frac{3L_m n}{4L_r}$)

- θ_r , ángulo entre los ejes magnéticos de las bobinas del rotor respecto a los ejes magnéticos de las bobinas del estator (rad)
- $i_{s\alpha}, i_{s\beta}$, componenetes de la intensidad del estator en los ejes $\alpha\beta$ (A)
- i_{sd}, i_{sq} , componenetes de la intensidad del estator en los ejes dq (A)
- $\lambda_{r\alpha}, \lambda_{r\beta}$, componenetes de los flujos del rotor en los ejes $\alpha\beta$ (wb)
- $\lambda_{rd}, \lambda_{rq}$, componenetes de los flujos en los ejes dq (wb)
- $V_{s\alpha}, V_{s\beta}$, componenetes del voltaje del estator en los ejes $\alpha\beta$ (V)

Los valores de los parámetros han sido los siguientes:

$R_r(\Omega)$	1.009	$R_s(\Omega)$	1.725
$L_r(H)$	0.1473	$L_s(H)$	0.1473
$L_m(H)$	0.1271	$J(kg \cdot m^2)$	0.04
$f_b(Hz)$	60	$f_d(Ns/m)$	0.75
n	4		