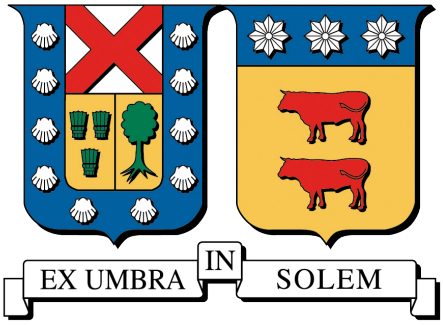
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE




# Pronóstico de fallas para mantenimiento predictivo usando metodologías de aprendizaje profundo supervisado

**Francisco Javier Araya López**

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL TELEMÁTICO

| | | |
|---|---|---|
| PROFESOR GUÍA | : | Mauricio Araya L. |
| PROFESOR CORREFERENTE | : | Nicolás Jara C. |

Diciembre 2020

# Agradecimientos

En primer lugar agradezco a las personas que me guiaron en mi decisión de entrar a esta universidad, en especial a mi profesor de enseñanza media Orlando Lara, ex-sansano el cual desinteresadamente me ofreció su ayuda cuando más la necesitaba.

También he de agradecer a las personas y amigos que conocí en este camino e hicieron más llevadero mi paso por la UTFSM. Sin sus palabras de aliento hubiese sido más difícil terminar esta etapa.

Además agradezco a mi profesor guía Mauricio Araya por orientar mi memoria de titulación y siempre estar disponible cuando necesitaba aclarar dudas o corregir este trabajo.

Por último y lo más importante es agradecer a mi familia, en especial a Margarita López mi madre por siempre creer en mi, animarme y desde pequeño enseñarme que el esfuerzo trae recompensas. Faltarían palabras para agradecer todo lo que has hecho por mi en esta vida.

# Pronóstico de fallas para mantenimiento predictivo usando metodologías de aprendizaje profundo supervisado

**Francisco Javier Araya López**

Memoria para optar al Título de Ingeniero Civil Telemático.

**Universidad Técnica Federico Santa María**

*Profesor Guía:* Mauricio Araya L.

Diciembre 2020

***Resumen***

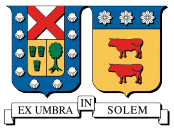
La predicción de fallas es una área importante a desarrollar por organizaciones que buscan maximizar el tiempo de producción de su maquinaria y/o minimizar los tiempos de mantenimientos (programados y no programados) [1]. Los costos de mantención pueden representar entre el 15 % y 60 % del valor de un bien producido dependiendo del área de la industria a la que pertenece la empresa productora. En países como EEUU un tercio de cada dólar gastado en mantenimientos se desperdicia por ser innecesarios o realizados incorrectamente. Lo anterior resalta un problema evidente de malgasto y pérdida de recursos.

Gracias al uso de sensores, es posible mejorar el monitoreo de condiciones de operación de componentes y/o sistemas que necesitan mantenimiento más allá de lo que la inspección visual permite. Ante una gran cantidad de sensores monitoreando el funcionamiento de un sistema, se hace necesario contar con un modelo que estime su degradación utilizando como entrada los datos entregados por estos sensores (features), para luego ejecutar un mantenimiento predictivo eficaz. El resultado más conocido del análisis y evaluación de los features en el modelo de degradación es llamado Remaining Useful Life (RUL).

Dado el problema anterior, en este trabajo de memoria se aborda el desarrollo de una solución utilizando arquitecturas de aprendizaje profundo supervisado para pronóstico de fallas mediante la estimación del RUL. Los modelos generados utilizando estas arquitecturas son entrenados usando el Turbofan Engine Degradation Simulation Dataset proporcionado por la NASA. Adicionalmente, se ejecutan distintas técnicas para mejorar las métricas de desempeño de estos modelos: preprocesamiento de datos, configuración de hiperparámetros y modificaciones a las arquitecturas mencionadas en el estado del arte.

Mediante el uso de JupyterLab, Google Colab, Keras, Python y librerias tales como Scikit-Learn, Pandas, Numpy y Matplotlib más el uso de CUDA se entrenan 32 modelos utilizando las distintas arquitecturas, datasets, configuraciones de hiperparámetros y time windows. Interpretando los resultados mediante las métricas de desempeño se observa que el desarrollo de la solución obtiene mejores indicadores que las arquitecturas descritas en el estado del arte. Por último, se realizan recomendaciones para el mantenimiento predictivo de acuerdo a las condiciones de operación de cada motor del Turbofan Engine Degradation Simulation Dataset.

**Palabras Clave.** Mantenimiento Predictivo, RUL, Turbofan Engine Degradation Simulation Dataset, Series de tiempo multivariadas, Aprendizaje Profundo, CNN, Dropout, Adam, RMSE, Keras, Python.





# Failure forecasting for predictive maintenance using supervised deep learning methodologies

**Francisco Javier Araya López**

Final project report towards the fulfillment of Ingeniero Civil Telemático (6 year program).

**Universidad Técnica Federico Santa María**

*Advisor:* Mauricio Araya L.

December 2020

***Abstract***

Failure prediction is an important area to develop by organizations seeking to maximize the production time of their machinery and/or minimize maintenance times (scheduled and unscheduled) [1]. Maintenance costs can represent between 15 % and 60 % of the value of a good produced depending on the area of the industry to which the producing company belongs. In countries like the US, a third of every dollar spent on maintenance is wasted because it is unnecessary or performed incorrectly. The foregoing highlights an obvious problem of waste and loss of resources.

Thanks to the use of sensors, it is possible to improve the monitoring of operating conditions of components and/or systems that need maintenance beyond what visual inspection allows. Faced with a large number of sensors monitoring the performance of a system, it is necessary to have a model that estimates its degradation using as input the data delivered by these sensors (features), to then execute effective predictive maintenance. The best known result of the analysis and evaluation of the features in the degradation model is called Remaining Useful Life (RUL).

Given the previous problem, this final project report addresses the development of a solution using supervised deep learning architectures for failure forecasting by estimating RUL. Models generated using these architectures are trained using the Turbofan Engine Degradation Simulation Dataset provided by NASA. Additionally, different techniques are executed to improve the performance metrics of these models: data preprocessing, hyperparameter configuration and modifications to the architectures mentioned in the state of the art.

Through the use of JupyterLab, Google Colab, Keras, Python and libraries such as Scikit-Learn, Pandas, Numpy and Matplotlib plus the use of CUDA, 32 models are trained using the different architectures, datasets, hyperparameter configurations and time windows. Interpreting the results through the performance metrics, it is observed that the development of the solution obtains better indicators than the architectures described in the state of the art. Finally, recommendations for predictive maintenance are made according to the operating conditions of each engine in the Turbofan Engine Degradation Simulation Dataset.

**Keywords.** Predictive Maintenance, RUL, Turbofan Engine Degradation Simulation Dataset, Multivariate time series, Deep Learning, CNN, Dropout, Adam, RMSE, Keras, Python.

# Glosario

**Python:** Lenguaje de programación interpretado y multiparadigma. Su filosofía hace hincapié en la legibilidad del código generado.

**Numpy:** Biblioteca de Python que da soporte para la creación de vectores y matrices multidimensionales de gran tamaño adicionando funciones para operar con ellas.

**Pandas:** Es una biblioteca de extensión de Numpy para manipulación y análisis de datos en el lenguaje de programación Python.

**Matplotlib:** Biblioteca de generación de gráficos en el lenguaje de programación Python.

**CUDA:** Plataforma de computación en paralelo desarrollada por NVIDIA para computación general en unidades de procesamiento gráfico (GPU).

**GPU:** Sigla en inglés usada para referirse a una unidad de procesamiento gráfico. Gracias a que posee una gran cantidad de procesadores es utilizada para acelerar el entrenamiento de modelos de aprendizaje profundo.

**CPU;** Sigla en inglés usada para referirse a una unidad central de procesamiento. Mediante la realización de operaciones aritméticas y lógicas es capaz de interpretar instrucciones de algún programa informático.

**Jupyter Lab:** Entorno que permite trabajar con documentos y archivos tales como Jupyter Notebooks, editores de texto y terminales.

**Google Colab:** Entorno cloud que permite ejecutar código y texto en un documento llamado Colab Notebook. Es posible insertar además imágenes, código HTML y Látex en ellos.

**NASA:** Agencia del gobierno estadounidense que realiza investigación aeroespacial y aeronáutica.

# Índice de Contenidos

# Índice de Tablas

# Índice de Figuras

# 1 | Introducción

## 1.1. Contexto

La predicción de fallas es una área importante a desarrollar por organizaciones que buscan maximizar el tiempo de producción de su maquinaria y/o minimizar los tiempos de mantenimientos (programados y no programados) [1]. Los costos de mantención pueden representar entre el 15 % y 60 % del valor de un bien producido dependiendo del área de la industria a la que pertenece la empresa productora. Encuestas respecto a efectividad de la gestión en el mantenimiento elaboradas en EEUU, indican que un tercio de cada dólar gastado en mantención se desperdicia debido a un mantenimiento innecesario o realizado incorrectamente. A modo de ejemplo, la industria estadounidense invierte aproximadamente $200 billones de dólares en mantenimiento al año. Es posible calcular que $66.6 billones de dólares corresponden a gastos de mantenimientos innecesarios o realizados incorrectamente. Este cálculo resalta un problema evidente de malgasto y pérdida de recursos.

Gracias al desarrollo de sensores y tecnología IoT de los últimos años [2], es posible mejorar el monitoreo de condiciones de operación de maquinaria, motores y componentes de sistemas que necesitan mantenimiento más allá de lo que la inspección visual permite. El añadir estos instrumentos impacta positivamente en reducir el número de mantenimientos innecesarios y las fallas inesperadas de componentes y sistemas.

Lo anterior introduce una complejidad más: al añadir más sensores a los sistemas, se aumenta el número de features (características) que deben ser supervisadas en tiempo real. Por ejemplo, en un sistema monitoreado por 100 sensores es difícil determinar la importancia relativa de cada uno y que rol tienen en la degradación del mismo. Evidentemente, se desea contar con un modelo que estime la degradación de estos sistemas utilizando como entrada estos features. El tratamiento de las problemáticas mencionadas se denomina mantenimiento predictivo.

## 1.2. Mantenimiento predictivo

El mantenimiento predictivo está definido como: *"Un mantenimiento basado en la condición llevado a cabo después de un pronóstico derivado del análisis y evaluación de los parámetros significativos de la degradación del ítem"* [3]. En detalle, el mantenimiento predictivo consiste en una mantención programada de acuerdo al análisis y evaluación de los features obtenidos del sistema (mediante sensores por ejemplo) dado un modelo de degradación obtenido previamente (ver Figura 1.1). El resultado más conocido del análisis y evaluación de los features en el modelo de degradación es llamado RUL (Remaining Useful Life).

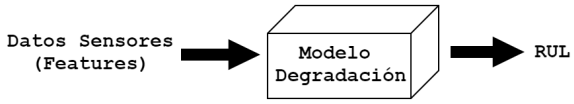



**Figura 1.1:** Esquema Features - Modelo degradación - RUL
(Fuente: Elaboración propia)

**Definición 1.2.1** (RUL)**.** Sea $T$ el tiempo de falla del sistema y suponga que el sistema ha sobrevivido hasta el tiempo $t$, entonces la variable aleatoria condicional:

$$X_n = T - t \qquad \text{cuando } T > t, \tag{1.1}$$

es el tiempo restante de falla o RUL del sistema [4].

En síntesis, el RUL describe cuantos ciclos de vida le restan a un componente y/o sistema desde su estado actual de degradación hasta que falle (ver Figura 1.2).

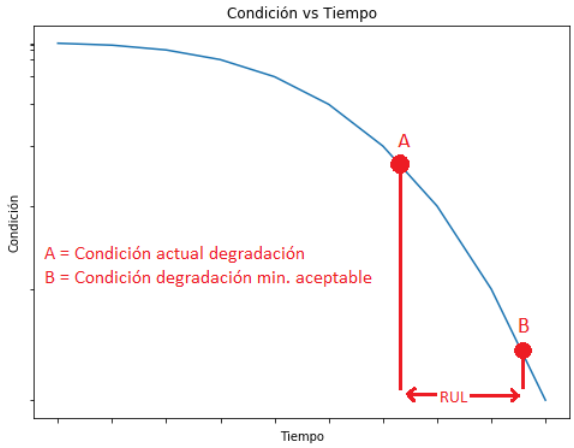



**Figura 1.2:** Cálculo RUL (T - t) con punto A = t y punto B = T
(Fuente: Elaboración propia basado en
https://medium.com/@RemiStudios/remaining-useful-life-in-predictive-maintenance-ffc91d7e4a97)

Para generar los modelos de degradación se pueden utilizar algoritmos de aprendizaje automático [5]. Luego, es posible obtener el RUL del componente o sistema mediante la evaluación de los features (obtenidos mediante sensores) en el modelo de degradación. Finalmente, interpretando el valor de RUL obtenido es posible ejecutar un mantenimiento predictivo eficaz. Todo lo anterior se puede realizar siempre y cuando se tengan a disposición datos históricos del sistema tales como datos de features y su respectivo RUL para entrenar el modelo supervisado de aprendizaje automático.

## 1.3. Aprendizaje Automático en Mantenimiento Predictivo

### 1.3.1. Conceptos

Un algoritmo de aprendizaje automático es un algoritmo que puede aprender de los datos [6]. Según [7] la definición de aprendizaje automático es la siguiente: *"Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas T y medida de rendimiento P, si su desempeño en tareas en T, medido por P, mejora con la experiencia E".*

En la definición anterior:

- **Tarea T:** Se refiere a los tipos de tareas que pueden ser resueltas mediante aprendizaje automático. Sea $\mathbf{x} = \{x_1, x_2, \dots, x_n\} \in \mathbb{R}^n$ una colección de features, diremos que $\mathbf{x}$ es una muestra o una entrada del modelo de aprendizaje automático donde cada valor $x_i$ es un feature. Por ejemplo, los datos entregados por sensores de un sistema en un instante (o ventana) de tiempo son una posible muestra o entrada a un modelo de aprendizaje automático. Algunas de las tareas que pueden ser resueltas con estos algoritmos son:
  - **Clasificación:** Se busca mapear cada entrada a una categoría dentro de un conjunto de k posibles. Para esto, el algoritmo busca una función $f(\mathbf{x}) : \mathbb{R}^n \to \{1, ..., k\}$, donde $y = f(\mathbf{x}) \quad y \in \{1, 2, ..., k\}$ es la predicción de la categoría a la cual corresponde $\mathbf{x}$.
  - **Regresión:** Este tipo de tarea es similar a la de clasificación, pero ahora cada entrada es mapeada a un valor numérico en $\mathbb{R}$. El algoritmo busca una función[1] $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$, donde $y = f(\mathbf{x}) \quad y \in \mathbb{R}$
  - **Detección de Anomalías:** El algoritmo busca agrupar muestras similares. Las muestras que no compartan similitudes con los grupos formados anteriormente son consideradas anómalas.
- **Medida de Rendimiento P:** Para evaluar que tan preciso es un modelo generado por un algoritmo de aprendizaje automático, se debe medir con un valor cuantitativo de rendimiento. Esta medida de rendimiento P generalmente está asociada al tipo de tarea T.

  Para tareas como clasificación se usa **accuracy** como medida P, el cual indica el porcentaje de entradas que al ser evaluadas en el modelo generan una predicción correcta. En tareas de regresión se usan medidas P como el **Mean Squared Error** (MSE) definido como:

$$\mathbf{MSE} = \frac{1}{N} \sum_{i=1}^{N} h_i^2 \tag{1.2}$$

  Donde $h_i$ es el error (distancia) entre el valor numérico estimado por el modelo y el valor numérico real asociado a la entrada $i$.
- **Experiencia E:** Los algoritmos de aprendizaje automático se dividen en dos grandes grupos: supervisados y no supervisados de acuerdo al tipo de experiencia que tienen en el proceso de aprendizaje (ver Figura 1.3). Se entiende como tipo de experiencia al dataset con el que el modelo de aprendizaje automático es entrenado. Un dataset es un conjunto de entradas o muestras, es decir:

$$\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(m)}\} \tag{1.3}$$

  Es un dataset compuesto de $m$ entradas con $n$ features cada una.

[1] Existen tareas de regresión multi-salida donde se cumple que $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$

- **Algoritmos supervisados:** El dataset está compuesto por features y un valor llamado label o target asociado a cada muestra. Estos algoritmos están principalmente asociados a tareas T de clasificación y regresión. En estos últimos años, los algoritmos de aprendizaje profundo supervisado (DNN, CNN, RNN) han ganado gran popularidad entre empresas tecnológicas y no tecnológicas volviéndose muy demandados para desarrollo de tecnologías asociadas a la Inteligencia Artificial [8].
- **Algoritmos no supervisados:** A diferencia de los algoritmos supervisados, el dataset solo está compuesto de features. Estos algoritmos buscan estructuras dentro de los datos de las muestras e intentan distinguir aquellas que tienen propiedades similares. Estos algoritmos están asociados a tareas T de detección de anomalías.

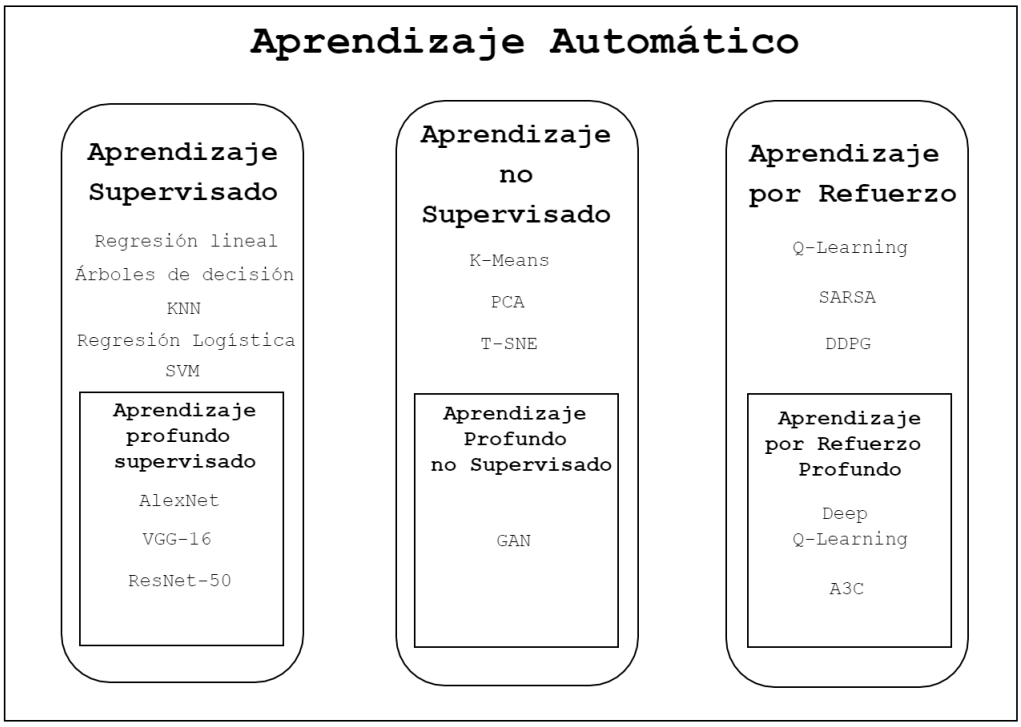



**Figura 1.3:** Arquitecturas y tipos de algoritmos de aprendizaje automático
(Fuente: Elaboración propia)

### 1.3.2. Entrenamiento y evaluación de modelos

Después de elegir el algoritmo adecuado acorde a la tarea que se quiere resolver, se procede a entrenar y evaluar el modelo generado [9]. Generalmente se divide el dataset en 3 conjuntos de datos: train, validation (dev) y test set. El modelo se entrena con los datos del train set, para luego validar su desempeño con los datos del validation set. Si el modelo logra un buen desempeño, se testea con los datos del test set.

El uso de un validation set se justifica para evitar el sobreajuste. Al tener este set disponible en la etapa de entrenamiento del modelo, es más rápido iterar en modificaciones a los hiperparámetros del algoritmo de aprendizaje automático. En aprendizaje profundo el entrenamiento de los modelos puede durar horas, por lo que solo tener un cálculo de medida de rendimiento mediante el test set al final de ese proceso es muy costoso en tiempo.

Se definen dos conceptos importantes tras el entrenamiento de un modelo de aprendizaje automático:

- **Subajuste:** Ocurre cuando el modelo generado en el proceso de entrenamiento no se ajusta correctamente a los datos del train set. Se dice que el modelo tiene un train error alto.
- **Sobreajuste:** En este caso los datos del test set no se ajustan bien al modelo generado en el proceso de entrenamiento, a diferencia de los datos del train set que se ajustan perfectamente. Se dice que el modelo no es capaz de generalizar para datos nuevos y tiene un test error alto.

  Estos conceptos se pueden apreciar de manera sencilla en la figura 1.4.

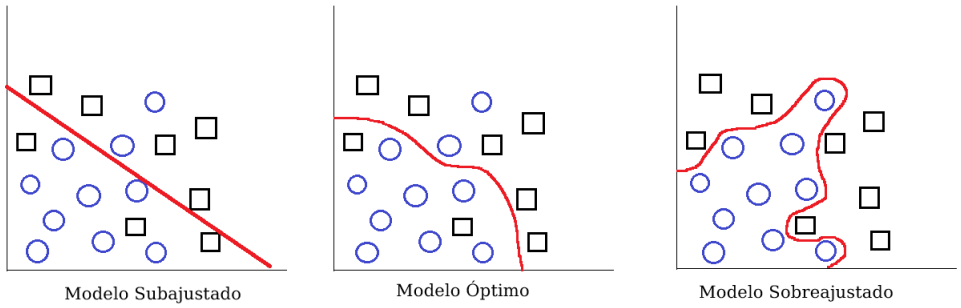



**Figura 1.4:** Modelo subajustado, modelo óptimo y modelo sobreajustado en aprendizaje automático
(Fuente: Elaboración propia)

En la figura 1.5 se visualiza el punto óptimo en el que un modelo de aprendizaje profundo tiene un valor bajo de train y test error dado un número de epochs, evitando así un subajuste y sobreajuste.

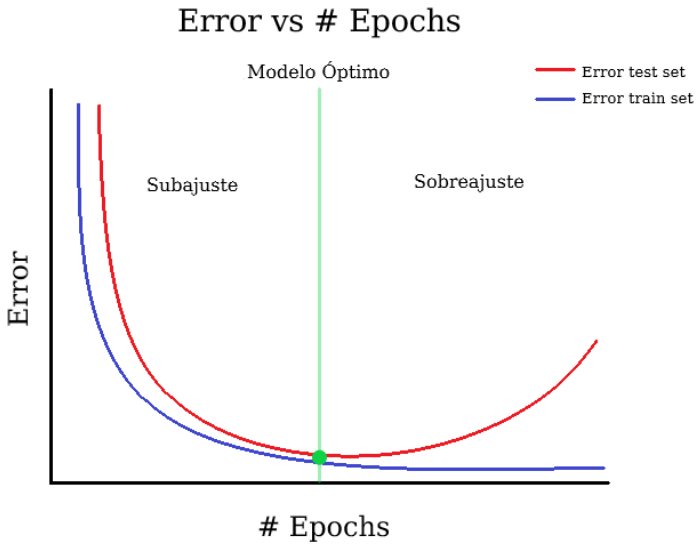



**Figura 1.5:** Gráfico Error vs número Epochs en modelo de aprendizaje profundo
(Fuente: Elaboración propia)

La evaluación final del modelo se hace con una medida de rendimiento P. En el caso de mantenimiento predictivo se desea encontrar una predicción del RUL, por lo que se usa como medida P el MSE ya que es una tarea T de regresión.

## 1.4. Preprocesamiento de datos

### 1.4.1. Series de tiempo irregularmente espaciadas en el tiempo

Los dataset de mantenimiento predictivo generalmente son series de tiempo multivariadas. Estas series en la mayoría de los casos son regularmente espaciadas en el tiempo, pero también existen aquellas que no lo son. Para series de tiempo irregularmente espaciadas en el tiempo, existen diferentes alternativas para tratar estos datos basadas en interpolaciones [10] tales como Simple exponential smoothing y métodos de Holt (Holt-Winters, Holt-Wright). El método de Holt-Wright [11] se define de la siguiente forma:

Sea $X_o...X_n$ datos irregularmente espaciados en tiempo $t_o, ..., t_n$ , $0 < a < 1$ y $0 < c < 1$ parámetros de suavizado y $q$ el promedio de los $\Delta t$ entre los tiempos de ocurrencia irregulares $t_o, ..., t_n$ se definen las siguientes ecuaciones:

$$V_n = V_{n-1}/(b_n + V_{n-1})$$

$$b_n = (1 - a)^{(t_n - t_{n-1})}$$

$$L_n = (1 - V_n)[L_{n-1} + (t_n - t_{n-1})M_{n-1}] + V_n X_n$$

$$M_n = (1 - U_n)M_{n-1} + U_n(L_n - L_{n-1})/(t_n - t_{n-1})$$

$$U_n = U_{n-1}/(d_n + U_{n-1})$$

$$d_n = (1 - c)^{(t_n - t_{n-1})}$$

$$V_o = 1 - (1 - a)^q$$

$$L_o = A$$

$$M_o = B$$

$$U_o = 1 - (1 - c)^q$$

Los valores $A$ y $B$ son el intercepto y la pendiente de la recta formada por los puntos $(t_o, X_o)$ y $(t_1, X_1)$. Tras utilizar el método de Holt-Wright se obtienen distintas rectas entre los datos irregularmente espaciados en el tiempo, las cuales permiten obtener un pronostico estimado de cada nuevo punto que se quisiera evaluar. Con lo anterior, es posible obtener una serie regularmente espaciada en el tiempo.

A continuación (ver Figura 1.6) se observa la aplicación del método de Holt-Wright:

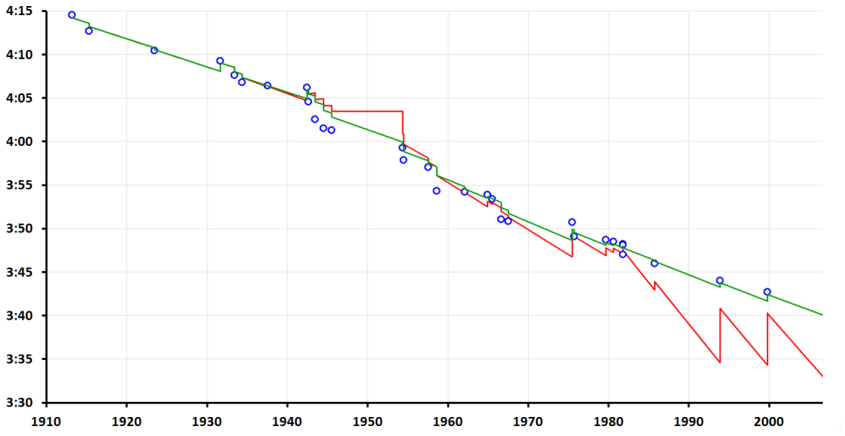



**Figura 1.6:** Método Holt-Wright (curva roja) vs método modificado de T. Hanzák (curva verde) aplicada a los récords mundiales de tiempo de una milla masculinos

(Fuente: T. Hanzák, Methods for periodic and irregular time series)

## 1.5. Objetivo General

Implementar una arquitectura de aprendizaje profundo para pronóstico de fallas mediante la estimación del Remaining Useful Life de un sistema.

### 1.5.1. Objetivos Especificos

- Comparar distintas arquitecturas de aprendizaje profundo usadas para tareas de regresión descritas en el estado del arte.
- Aplicar estas arquitecturas de aprendizaje profundo a un ejemplo de mantenimiento predictivo de un motor de avión usando los datos entregados por distintos sensores anclados a este.
- Determinar cómo afecta a la solución los diferentes hiperparámetros, optimizadores, dimensiones de las entradas y el preprocesamiento del conjunto de datos al desempeño de las arquitecturas.

# 2 | Aprendizaje Profundo Supervisado

## 2.1. Redes Neuronales

Una red neuronal artificial es un conjunto de múltiples capas de neuronas interconectadas entre si. Cada neurona recibe información de la capa anterior y la utiliza para realizar una transformación. La salida de estas neuronas son traspasadas a la siguiente capa de neuronas, así la red va aprendiendo en cada una de estas capas representaciones más complejas. La capa de salida entrega valores distintos dependiendo del objetivo de la red: 0/1 dependiendo si el objetivo fue usar la red para clasificación binaria, valores continuos si el objetivo era obtener un modelo de regresión, o valores discretos si el objetivo era construir un modelo de clasificación multiclases.

## 2.2. Neurona

Una neurona es la unidad más básica de las redes neuronales (ver Figura 2.1). Sea $x_{n\times 1}$ un vector de entrada, $w^T_{1\times n}$ una matriz de pesos transpuesta, $b$ un sesgo escalar y $f$ una función de activación, la salida $y$ de la neurona se obtiene mediante:

$$y = f(\sum_{i=1}^{n} w_i \cdot x_i + b) \tag{2.1}$$

En representación matricial, lo anterior se escribe de la forma:

$$\underbrace{\begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}}_{w^T_{1\times n}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}}_{x_{n\times 1}} + b = z \rightarrow f(z) = y \tag{2.2}$$

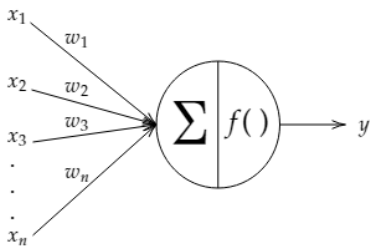



**Figura 2.1:** Representación neurona con $n$ entradas, $n$ pesos y función de activación $f$
(Fuente: Elaboración propia)

## 2.3. Modelo red neuronal

Una red neuronal está conformada por capas que agrupan neuronas (ver Figura 2.2). Cada capa luego del vector de entrada se considera una capa oculta salvo la última capa de salida. Para la primera capa oculta[2] y sea $j = 1, 2, ..., m$ la etiqueta de una neurona dentro de la capa, se tiene la salida $y_j^{[1]}$ como sigue:

$$y_j^{[1]} = f(\sum_{i=1}^{n} w_{ji} \cdot x_i + b_j^{[1]}) \tag{2.3}$$

En representación matricial, lo anterior se escribe de la forma:

$$\underbrace{\begin{bmatrix} w_{j1} & w_{j2} & \cdots & w_{jn} \end{bmatrix}}_{w_{1\times n}^{\top} = W_j^{[1]^T}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}}_{x_{n\times 1}} + b_j^{[1]} = z_j^{[1]} \rightarrow f(z_j^{[1]}) = y_j^{[1]} \tag{2.4}$$

Con la primera capa oculta conformada por $m$ neuronas, se pueden representar sus salidas de la siguiente forma:

$$\underbrace{\begin{bmatrix} W_1^{[1]^T} \\ W_2^{[1]^T} \\ \cdots \\ W_m^{[1]^T} \end{bmatrix}}_{W^{[1]}{}_{m\times n}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}}_{x_{n\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \cdots \\ b_m^{[1]} \end{bmatrix}}_{b^{[1]}{}_{m\times 1}} = \underbrace{\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ \cdots \\ z_m^{[1]} \end{bmatrix}}_{z^{[1]}{}_{m\times 1}} \rightarrow \underbrace{\begin{bmatrix} f(z_1^{[1]}) = y_1^{[1]} \\ f(z_2^{[1]}) = y_2^{[1]} \\ \cdots \\ f(z_m^{[1]}) = y_m^{[1]} \end{bmatrix}}_{y^{[1]}{}_{m\times 1}} \tag{2.5}$$

Para las siguientes capas con $k = 2, 3, ..., l$, se obtiene la salida $y_j^{[k]}$ como sigue:

$$y_j^{[k]} = f(\sum_{i=1}^{o} w_{ji} \cdot y_i^{[k-1]} + b_j^{[k]}) \tag{2.6}$$

La representación matricial de las salidas de la capa oculta $k$ es de la forma:

$$W^{[k]} y^{[k-1]} + b^{[k]} = z^{[k]} \rightarrow f(z^{[k]}) = y^{[k]} \tag{2.7}$$

Siendo:

- $W^{[k]}$: Matriz de pesos de la capa oculta $k$.
- $y^{[k-1]}$: Matriz de salidas de neuronas de la capa oculta $k-1$.
- $b^{[k]}$ : Matriz de sesgos de la capa oculta $k$.
- $y^{[k]}$ : Matriz de salidas de neuronas de la capa oculta $k$.

El número de capas ocultas, neuronas en cada capa oculta, entradas, inicialización de pesos y elección de funciones de activación depende de la arquitectura que requiera el usuario.

[2]La notación de corchetes $[i]$ se refiere al número de capa o capa oculta $i$.

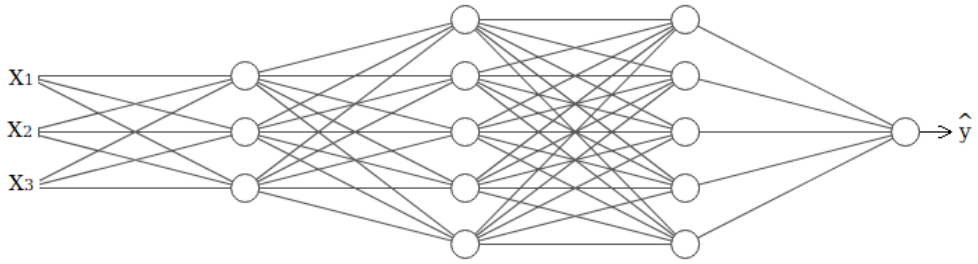



**Figura 2.2:** Ejemplo de representación de red neuronal multicapa (fully-connected) con 3 entradas, 3 capas ocultas y una capa de salida
(Fuente: Elaboración propia)

### 2.3.1. Funciones de activación

Las funciones de activación más usadas son Sigmoide, Tanh y ReLU. Se definen de la siguiente forma:

- Sigmoide: Sea $f : \mathbb{R} \to ]0, 1[$ se define la función Sigmoide como

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

La gráfica de la función Sigmoide se observa en la figura 2.3.

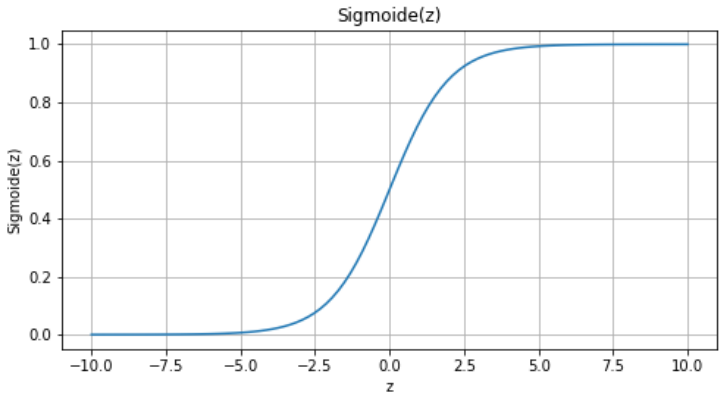



**Figura 2.3:** Función Sigmoide
(Fuente: Elaboración propia)

- Tanh: Sea $f : \mathbb{R} \rightarrow ]-1, 1[$ se define la función Tanh como

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

La gráfica de la función Tanh se observa en la figura 2.4.

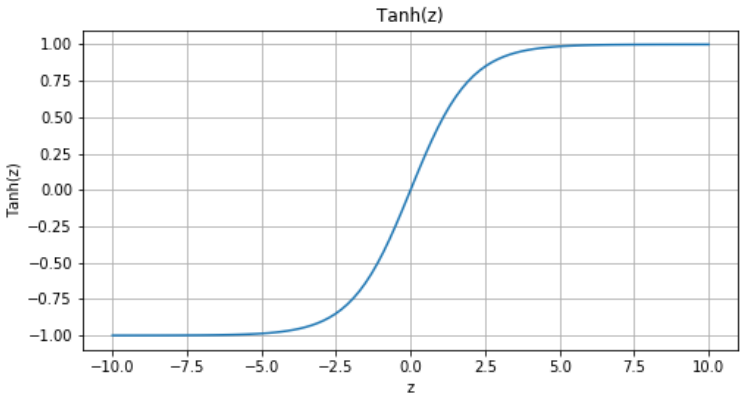



**Figura 2.4:** Función Tanh
(Fuente: Elaboración propia)

- ReLU: Sea $f : \mathbb{R} \rightarrow [0, \infty[$ se define la función ReLU como

$$f(z) = max(0, z) \tag{2.10}$$

La gráfica de la función ReLU se observa en la figura 2.5.

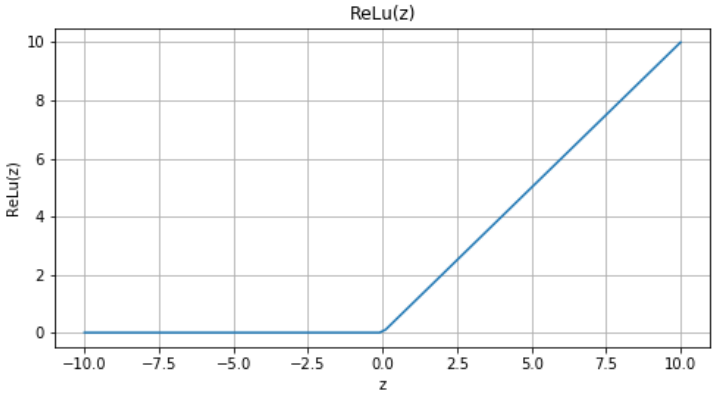



**Figura 2.5:** Función ReLU
(Fuente: Elaboración propia)

## 2.4. Convolutional Neural Networks (CNN)

Las redes neuronales convolucionales (CNN) son un tipo de red neuronal utilizada preferentemente en visión por computador, donde la transformación de datos se da gracias a la convolución entre un kernel (o filtro) y un sub-conjunto del conjunto de datos de entrada. Esta operación recorre todo el conjunto de entrada para generar una salida. La convolución en CNN entre una imagen $I$ de dimensiones $h$ (altura), $w$ (ancho) y un kernel $K$ se define de la siguiente manera:

$$C = (I * K)(i, j) = \sum_{m=1}^{h} \sum_{n=1}^{w} I(i + n, m + j)K(n, m) \tag{2.11}$$

Al realizar una operación de convolución, se puede observar que la salida resultante es de menor dimensión que la imagen de entrada (ver Figura 2.6), esta operación se dice que ocupa **valid padding**. Para preservar la dimensión, se utiliza la técnica de rellenar con ceros (zero padding).

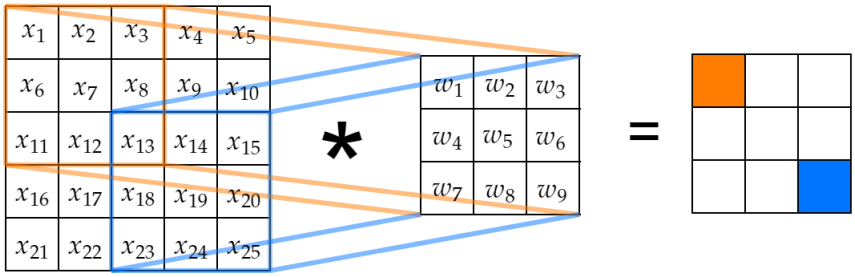



**Figura 2.6:** Representación proceso de convolución entre una matriz de entrada (Imagen) y kernel
(Fuente: Elaboración propia)

La técnica de zero padding consiste en rellenar con ceros un determinado número de nuevas columnas y filas alrededor de la imagen de entrada para así obtener una salida con la misma dimensión que esta. Al realizar una operación de convolución se puede ocupar **same padding**, técnica la cual ocupa zero padding para preservar la dimensión de los datos de entrada en la salida de la capa de convolución. Además, la técnica de **zero padding** permite que los valores de los píxeles en los extremos de la imagen aporten mayor información a la convolución resultante.

Sea la imagen de entrada de dimensiones $w \times h$, con padding $p$ y utilizando un kernel o filtro de dimensión $f$, la salida resultante tendrá dimensiones:

$$C_{(w+2p-f+1)\times(h+2p-f+1)} \tag{2.12}$$

### 2.4.1. Pooling layers

Además de las capas de convolución, en CNN existen otras capas que permiten reducir dimensionalidad en las capas internas de la red. Un ejemplo de ellas son las capas de agrupamiento o pooling layers, las cuales extraen características específicas de sus datos de entrada.

Las dos capas más utilizadas de este tipo son:

- **Max-Pooling:** Extrae el valor máximo de una sub-muestra de un espacio de características para así formar un nuevo espacio de características reducido (ver Figura 2.7).

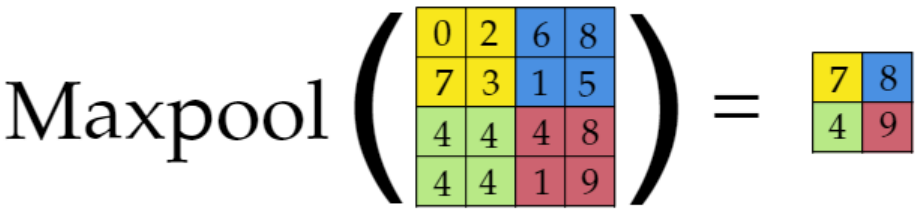



**Figura 2.7:** Representación proceso de Max-pooling
(Fuente: Elaboración propia)

- **Average-Pooling:** Es idéntico a Max-Pooling, con la diferencia de que extrae el valor promedio de una submuestra del espacio de características y forma con ello un nuevo espacio de características reducido (ver Figura 2.8).

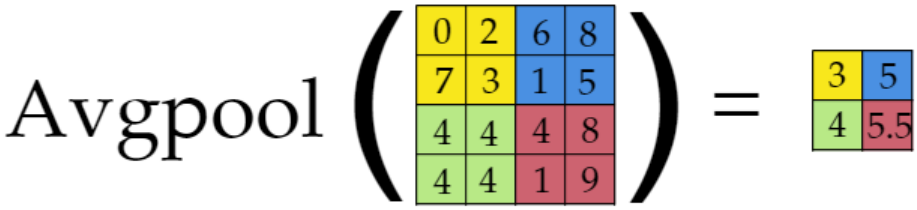



**Figura 2.8:** Representación proceso de Average-pooling
(Fuente: Elaboración propia)

## 2.5. Backpropagation

Se define $\{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), ..., (\mathbf{x}^{(m)}, y^{(m)})\}$ como el train set de un dataset donde cada muestra de $\mathbf{X}$ tiene asociado su valor real $y$. Después de calcular las salidas $y^{[k]}$ en cada capa oculta (forward-propagation) para cada muestra del train set, se procede a calcular el error entre el valor estimado o predecido por la red neuronal $\hat{y}^{(i)}$ y el valor real $y^{(i)}$ mediante una función de pérdida o Loss Function $L(\hat{y}, y)$. El promedio de la suma de estos errores es conocida como función de costo o Cost Function $J$ definida como:

$$J = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) \tag{2.13}$$

Dado que se desea que el modelo entrenado genere predicciones precisas, se espera minimizar la función $J$ (y los errores $L(\hat{y}, y)$). Usando el algoritmo de gradiente descendiente se actualizan las matrices de pesos y de sesgo en cada capa de la red:

$$W^{[k]} = W^{[k]} - \alpha \frac{\partial J}{\partial W^{[k]}} \tag{2.14}$$

$$b^{[k]} = b^{[k]} - \alpha \frac{\partial J}{\partial b^{[k]}} \tag{2.15}$$

donde $\alpha$ es conocida como tasa de aprendizaje o learning rate. Todo lo definido anteriormente se conoce como el proceso de back-propagation de una red neuronal.

Se define un epoch como el paso de todo el train set en el entrenamiento de un modelo de aprendizaje profundo. En el caso de gradiente descendiente un epoch equivale a una actualización de la Función de Costo $J$ y posterior back-propagation de la red. El proceso de aprendizaje de una red neuronal está asociado a la actualización de las matrices (2.14) y (2.15). En cada epoch las matrices se optimizan y el modelo genera predicciones más precisas con errores $L(\hat{y}, y)$ cada vez más pequeños. Se debe tener en cuenta que el número de epochs usados en el proceso de entrenamiento no sobreajuste el modelo generado como se observa en la figura 1.5.

## 2.6. Optimizadores

Existen más optimizadores como alternativa al algoritmo de gradiente descendiente. Algunos de estos son:

- **Mini-batch gradient descent:** En este algoritmo [12] el training set se divide en mini-batches de igual tamaño. Se realiza el proceso de forward-propagation y back-propagation en la red neuronal con cada mini-batch. Sea $B$ el tamaño de un mini-batch y $m$ el número de muestras del training set. Con $m > B$ se tiene:

$$\underbrace{\{(\mathbf{x}^{(1)}, y^{(1)}), ..., (\mathbf{x}^{(B)}, y^{(B)})}_{1\, mini-batch}, ... \underbrace{|..., (\mathbf{x}^{(m)}, y^{(m)})\}}_{\lceil \frac{m}{B} \rceil mini-batch} \tag{2.16}$$

El cálculo de $J$ queda definido como:

$$J = \frac{1}{B}\sum_{i=1}^{B} L(\hat{y}^{(i)}, y^{(i)}) \tag{2.17}$$

(2.14) y (2.15) se actualizan en el entrenamiento del modelo con cada mini-batch. El uso de mini-batches acelera el entrenamiento de los modelos de aprendizaje profundo supervisado si es que el tamaño $m$ del dataset es demasiado grande, pero la optimización de $J$, $W^{[k]}$ y $b^{[k]}$ son más imprecisas, dado que estas se actualizan en cada iteración con un conjunto de datos más pequeño comparado a la optimización por gradiente descendiente.

- **Stochastic Gradient Descent (SGD):** Es un caso particular de Mini-batch gradient descent [12] en el cual el tamaño del minibatch es $B = 1$. Donde se calcula para cada muestra:

$$J = L(\hat{y}^{(i)}, y^{(i)}) \tag{2.18}$$

El proceso de aprendizaje (forward-propagation y back-propagation) y la actualización de $W^{[k]}$ y $b^{[k]}$ se realiza solo con el error asociado a una muestra a la vez del train set.

En SGD (2.14) y (2.15) quedan definidos como:

$$W^{[k]} = W^{[k]} - \alpha\frac{\partial L}{\partial W^{[k]}} \tag{2.19}$$

$$b^{[k]} = b^{[k]} - \alpha\frac{\partial L}{\partial b^{[k]}} \tag{2.20}$$

El uso de SGD acelera aún más el entrenamiento y actualización de las matrices $W^{[k]}$ y $b^{[k]}$, pero la búsqueda del óptimo en cada iteración es más imprecisa que en mini-batch gradient descent.

- **Adam:** Es un algoritmo de optimización de aprendizaje profundo [13] que posee propiedades tales como: sencillo de implementar, computacionalmente eficiente, requiere un uso pequeño de memoria y es adecuado para problemas grandes en número de entradas e hiperparámetros. Este algoritmo se utiliza en dos experimentos [13] : entrenando una red fully-connected con el MNIST dataset y entrenando una CNN con el CIFAR-10 dataset. Comparativamente con otros algoritmos de optimización como RMSProp, SGDNesterov, AdaGrad y AdaDelta obtiene un menor valor de training cost (error) en cada epoch permitiendo una convergencia más rápida de $J$.

# 3 | Estado del Arte

El aprendizaje profundo ha tomado gran protagonismo durante estos últimos años debido al incremento en capacidad de cómputo tanto de CPUs y GPUs. Estos algoritmos pueden ser usados para resolver problemas como: regresión, clasificación, predicción de series de tiempo, detección de objetos, etc.

Diversos autores han utilizado el enfoque de aprendizaje profundo para solucionar problemas referentes a la estimación del RUL. Zhigang Tian [14] utilizó una ANN para estimar un porcentaje de vida de rodamientos el cual le permite calcular el RUL. Su mayor problema es el ruido al que estan sometidos los sensores de medición, por lo que usa una función generalizada de la distribución de Weibull para mejorar la representación de los valores de entrada a la ANN. El autor concluye que siempre es necesario un preprocesamiento de los datos antes de entrenar una red neuronal. Por último, para obtener una mejor predicción del RUL además de usar el historial de fallas utiliza el historial de suspensión (de la bomba de agua a la que están anexados los rodamientos) rescatando los datos entregados por los sensores.

Li, Ding y Sun [15] a su vez proponen una arquitectura basada en una red neuronal convolucional (CNN) para predecir el RUL del Turbofan Engine Degradation Simulation Dataset. Su arquitectura utiliza filtros (kernels) de 10x1 en las primeras 4 capas convolucionales de la red emulando la convolución en 1 dimensión y usan a su vez como funciones de activación tangentes hiperbólicas (*tanh*). Luego utilizan otra capa de convolución donde se usa un filtro de 3x1 con función de activación *tanh*. La capa que precede lo anterior es Flatten, donde se redimensiona la matriz de características anterior a un vector de 1 columna, además de aplicar Dropout (una alternativa a la regularización L2) para evitar que el modelo se sobreajuste. En la siguiente etapa se usa una capa fully-connected donde cada neurona se conecta a cada salida de la capa anterior y las salidas de esta capa son conectadas completamente a la capa siguiente. Por último, la capa final es una neurona que toma todas las salidas de la capa anterior como inputs, añade los pesos y mediante una función de activación hace la regresión y posterior predicción del RUL. Comparan su método respecto a otras arquitecturas de redes neuronales como DNN, RNN y LSTM (un tipo de RNN) obteniendo resultados sobresalientes.

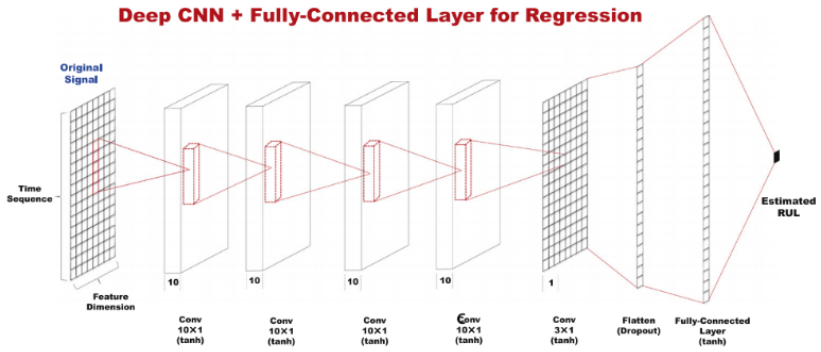



**Figura 3.1:** Arquitectura CNN propuesta por Li, Ding y Sun
(Fuente: X. Li, Q. Ding, J. Sun, Remaining useful life estimation in prognostics using deep convolution neural networks)

Babu, Zhao y Li [16] también proponen una CNN para el cálculo del RUL del Turbofan Engine Degradation Simulation Dataset. Su arquitectura y enfoque es parecido al paper anterior pero utilizan capas de Pooling las cuales tienen por objetivo reducir la dimensionalidad de las salidas de las capas convolucionales, resaltando características relevantes (valores máximos si se usa max-pooling o promedio si se usa una capa de average-pooling). La red queda conformada de la siguiente manera: $Inputs \rightarrow Conv \rightarrow Pooling \rightarrow Conv \rightarrow Pooling \rightarrow Fully - Connected \rightarrow$ Estimación de RUL mediante una capa de una sola neurona al igual que [15]. Comparan sus resultados respecto a otros algoritmos de aprendizaje automático tales como Multi-layer Perceptron (MLP), Support Vector Regression (SVR) y Relevance Vector Regression (RVR) obteniendo valores de Root Mean Square Error (RMSE) más bajos que estos.

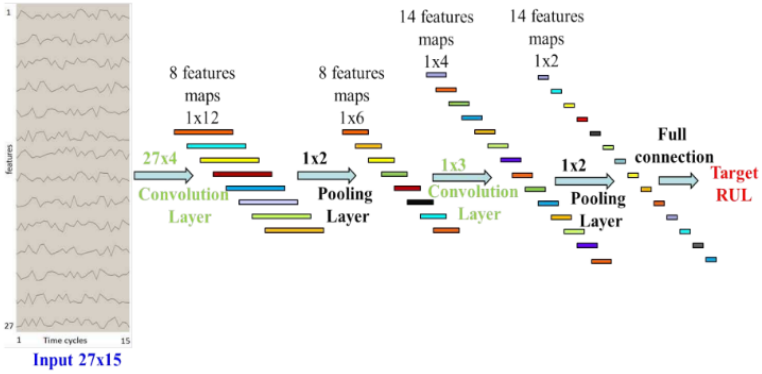



**Figura 3.2:** Arquitectura CNN propuesta por Babu, Zhao y Li
(Fuente: G. Babu, P. Zhao, X.-L. Li, Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life)

Por último, una opción a las CNN en dos dimensiones que son ampliamente utilizadas en visión por computador, Chollet [17] recomienda el uso de CNN de 1 dimensión para datos en secuencia y series de tiempo. Destaca sus ventajas como el costo computacional respecto a utilizar una RNN. Además, destaca que estas redes pueden reconocer patrones en secuencias cortas (menos de 1000 ciclos) y que las capas usadas en CNN de dos dimensiones como Pooling pueden ser utilizadas también en 1 dimensión. Es decir, al utilizar max-pooling en una sub-secuencia, se extrae el mayor valor de esta de manera análoga a un max-pooling de dimensión 2. El autor advierte que se necesita una RNN adicional a la CNN para tratar secuencias muy largas (sobre 1000 ciclos).

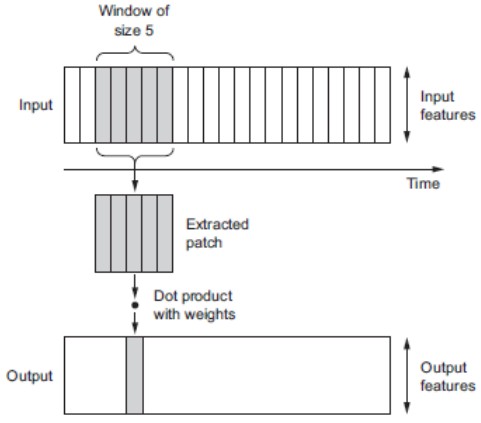



**Figura 3.3:** Como trabaja una convolución en 1D
(Fuente: Chollet, Deep Learning with Python)

# 4 | Datasets Mantenimiento predictivo

Se presentan a continuación dos datasets en los cuales es necesario predecir el RUL. El primero pertenece a la máquina encintadora Vega Shrink-wrapper y el segundo a un motor de avión.

## 4.1. Vega Shrink-wrapper Dataset

Esta máquina permite realizar completamente el proceso de encintado de packs de latas y botellas. Es decir, es capaz de agrupar botellas, encintarlas en un pack y luego sellarlas. En el módulo de encintado, existe una cuchilla para cortar el tamaño de película plástica necesaria para el encintado. Se espera predecir la falla (mediante la predicción del RUL) en la cuchilla de la máquina. Dado que esta se encuentra confinada en un lugar cerrado, es imposible inspeccionar visualmente su estado.

Cada archivo del dataset[3] contiene 8 features distintos que pueden ser medidos en el módulo donde se encuentra la cuchilla. Cada fila del dataset (muestras) corresponde a un instante de tiempo (regularmente espaciada en el tiempo cada 4 [ms] durante 8 [s]) en el cual se obtuvieron los respectivos valores de los campos mostrados a continuación:

| Variable | Descripción | Valores |
|---|---|---|
| timestamp | Estampa de tiempo [s] | [-0.192000, 8.187999] |
| motor_torque | Torque del motor de la cuchilla [nM] | [-2.843960, 0.632072] |
| pcut_pos_error | Error de posición instantáneo entre el punto de ajuste del generador de trayectoria y la posición real del encoder de posición del motor | [-0.599982, 0.346109] |
| blade_position | Posición de la cuchilla [mm] | [-884747, -783498] |
| blade_speed | Velocidad de la cuchilla | [-954.43, 3570.55] |
| film_unwinder_pos | Posición del desenrollador de la película plástica [mm] | [35096, 113597] |
| speed_film_unwinder | Velocidad del desenrollador de película plástica | [-25.04, 2742.25] |
| pfilm_pos_error | Error de posición instantáneo entre el punto de ajuste del generador de trayectoria y la posición real del encoder de posición del desenrollador de película plástica | [-0.086, 0.962] |

**Figura 4.1:** Descripción features Vega Shrink-wrapper Dataset
(Fuente: Elaboración propia)

[3] https://www.kaggle.com/inIT-OWL/one-year-industrial-component-degradation [Última consulta: 03-08-2020]

Los 518 archivos del dataset tienen el formato MM-DDTHHMMSS_NUM_modeX.csv, donde MM es el mes que va del 1 al 12, DD es el día del mes, HHMMSS es la hora de inicio del día de la grabación, NUM es el número de muestra y X es un modo que va del 1 al 8. Además, se cuenta con otro dataset[4] que contiene 3 archivos de procesos de cuchillas nuevas y otros 3 archivos de procesos de cuchillas gastadas con los mismos features y cantidad de filas.

### 4.1.1. Preprocesamiento del dataset:

Este dataset no cuenta con un feature específico que contenga el RUL en cada ciclo de uso de la cuchilla. Por lo que, es necesario el uso de técnicas de aprendizaje automático para encontrar una función que describa la degradación de esta. Al obtener la función de degradación, es posible obtener el RUL de cada ciclo y anexarlo al dataset. Teniendo lo anterior, el dataset estaría listo para que mediante regresión (con algún modelo de aprendizaje automático supervisado) se logre predecir el RUL de la cuchilla. Los pasos para encontrar una función de degradación con el Vega Shrink-wrapper Dataset son los siguientes:

- **Extracción de características:** Cada archivo del dataset (518 archivos de procesos de corte durante el año más los 6 archivos de procesos de corte con cuchillas nuevas y gastadas) es sometido a una extracción de características como lo realizado por Caesarendra y Tjahjowidodo [18]. Se obtiene el promedio, varianza, simetría, kurtosis y factor de cresta de cada feature (excluyendo el timestamp) en cada archivo. Por lo que, nuestro dataset luego de este proceso consta de 524 registros de datos con 42 features cada uno.
- **Reducción de dimensionalidad:** Al tener 42 features nuevos, se desea reducir el número de estos para simplificar el análisis posterior de los datos sin perder mucha información. Una técnica para lograr esto es PCA (Principal Component Analysis), la cual mediante una transformación lineal aplicada a todos los features es capaz de crear nuevas dimensiones ordenadas de mayor varianza (componentes principales) a menor varianza. Se observa en la figura 4.2 que con 8 componentes se alcanza una explicación de varianza acumulada mayor a 80 %.

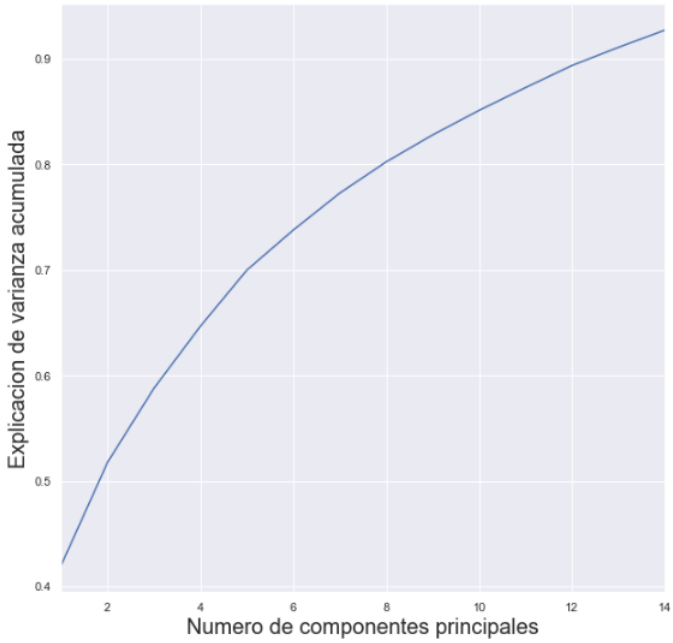



**Figura 4.2:** Gráfico varianza acumulada vs componentes principales Vega shrink-wrapper Dataset
(Fuente: Elaboración propia)

[4] https://www.kaggle.com/inIT-OWL/vega-shrinkwrapper-runtofailure-data [Última consulta: 03-08-2020]

- **Clustering:** Se utiliza T-SNE para buscar similitudes entre las muestras del dataset. T-SNE es una técnica que permite visualizar cada muestra del dataset en una región de dos o tres dimensiones. Cada punto generado se agrupa junto con otros puntos similares en clusters, lo que permite una óptima visualización de los datos.

  Se aprecia en la figura 4.3 que los puntos generados a partir de los registros de cuchillas nuevas y gastadas están separadas de los registros de las cuchillas correspondientes al año de uso. Esto se debe a que el comportamiento de la cuchilla es fuertemente dependiente del modo (de uso) de la Vega shrink-wrapper, por lo que obtener una curva de degradación es muy difícil.


**Figura 4.3:** Gráfico T-SNE aplicado a componentes principales Vega shrink-wrapper Dataset
(Fuente: Elaboración propia)

Sumado al problema anterior, solo se cuenta con 6 registros de cuchillas buenas y gastadas versus 518 sin etiquetar para enfrentar el problema con un modelo supervisado o semi-supervisado de clasificación, por lo que se obtendría un modelo subajustado. Por último, se podría construir un modelo no supervisado de degradación pero este debe ser revisado por expertos que certifiquen su validez. Se puede concluir que no todo dataset puede ser usado para obtener el RUL si es que no existe una calidad mínima de la supervisión como sucede en este dataset (dado que se deseaba encontrar un modelo de degradación que lograra mapear los features a un valor de RUL) ya que el uso de la máquina encintadora (modos) tiene un comportamiento estocástico y es muy difícil determinar que es lo que se desea aprender, validar un modelo y confiar en una predicción dada por el mismo. Por simplicidad y todos los problemas mencionados anteriormente se opta por descartar este dataset.

## 4.2. Turbofan Engine Degradation Simulation Dataset

Este dataset[5] refleja la degradación de un motor turbofan mediante series de tiempo multivariadas. Fue simulado con un software de la NASA llamado C-MAPPS (Comertial - Modular Aero-Propulsion System Simulation). Cada serie de tiempo corresponde a un motor distinto, es decir que el dataset puede considerarse un lote de motores del mismo tipo. Cada motor empieza en una condición inicial distinta desconocida para el usuario pero esta es considerada como normal (punto A de la Figura 1.2 por ejemplo).

En los conjuntos de datos de entrenamiento (train_FD00X.txt con $1 \leq X \leq 4$) el motor opera normalmente al inicio de la serie de tiempo, pero cae en una falla al final de esta. En el caso de los conjuntos de datos de prueba (test_FD00X.txt con con $1 \leq X \leq 4$) el motor opera normalmente y la serie de tiempo termina antes de que ocurra la falla del motor. El objetivo es predecir el RUL de cada motor en el conjunto de prueba. Para verificar las predicciones, se adjunta un vector de los RUL reales de cada motor en estos conjuntos.

En total cada dataset consiste en 26 columnas que corresponden a features de los motores, y cada fila es un ciclo (regularmente espaciado en el tiempo) de funcionamiento del respectivo motor. En detalle cada feature corresponde a:

- **n_engine:** Indica el número del motor simulado. Cada número puede ser considerado como un motor independiente de los demás.
- **cycle:** Ciclo de funcionamiento del motor respectivo.
- **opset_1 - opset_2 - opset_3:** Reflejan una condición operacional del motor. El significado de cada condición operacional es desconocido para el usuario.
- **sens_1 - ... - sens_21 :** Corresponden a sensores que miden 21 features distintos dentro del motor. El significado de cada sensor también es desconocido para el usuario.

En total el Turbofan Engine Degradation Simulation Dataset consiste en 4 datasets con diferentes condiciones operacionales, cantidad de motores simulados y número de registros. El detalle a continuación en la siguiente tabla:

**Tabla 4.1:** Detalle Turbofan Engine Degradation Simulation Dataset

| Dataset | N° Motores train set | N° Motores test set | Condiciones Operación | Modos de falla |
|---|---|---|---|---|
| FD001 | 100 | 100 | 1 | 1 |
| FD002 | 260 | 259 | 6 | 1 |
| FD003 | 100 | 100 | 1 | 2 |
| FD004 | 248 | 249 | 6 | 2 |

Además, a diferencia del Vega Shrink-wrapper Dataset es posible obtener el RUL de los motores gracias a que se informó anteriormente cuando un motor falla (en el caso de los archivos train_FD00X.txt un motor tiene RUL = 0 cuando la serie de tiempo finaliza y en los archivos test_FD00X.txt se cuenta con un vector de RUL al finalizar cada serie).

[5] https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/#turbofan [Última consulta: 03-08-2020]

### 4.2.1. Análisis descriptivo del dataset

Se utiliza PCA en cada uno de los 4 dataset. Se desea visualizar si los datos tienen un comportamiento lineal o no lineal graficando la primera componente principal de los features versus el RUL asociado a ellos en cada ciclo. En las figuras 4.4 a la 4.7 se visualiza un comportamiento no lineal de los datos, por lo que usar un modelo no lineal es adecuado para solucionar el problema de predecir el RUL de este dataset. Además, se observa que la distribución de los datos en los 4 datasets son representaciones complejas y distintas, por lo que un modelo que logre capturar estas complejidades también es adecuado. Dado lo anterior, utilizar un algoritmo de aprendizaje profundo supervisado nos garantiza obtener modelos no lineales y que logren capturar las representaciones complejas dentro de los datasets. Esto es posible mediante el aumento de la capacidad del modelo obtenido al añadir capas ocultas en el caso de redes neuronales fully-connected y/o al añadir capas convolucionales junto con más filtros en cada capa de una CNN por ejemplo.

Por otra parte, es posible observar cada una de las seis condiciones de operación en cada curva de la gráfica de los dataset FD002 y FD004. En el caso de los datasets FD001 y FD003 se observa solo una curva en la gráfica, asociada a la única condición de operación existente en ellos.

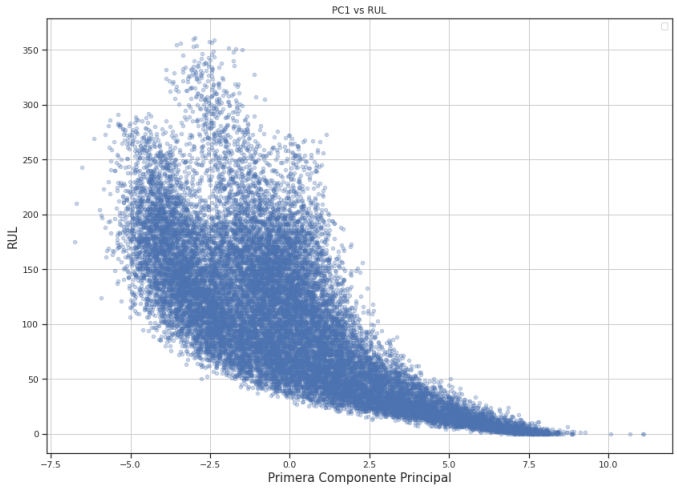



**Figura 4.4:** Gráfico primera componente principal versus RUL del train set FD001
(Fuente: Elaboración propia)

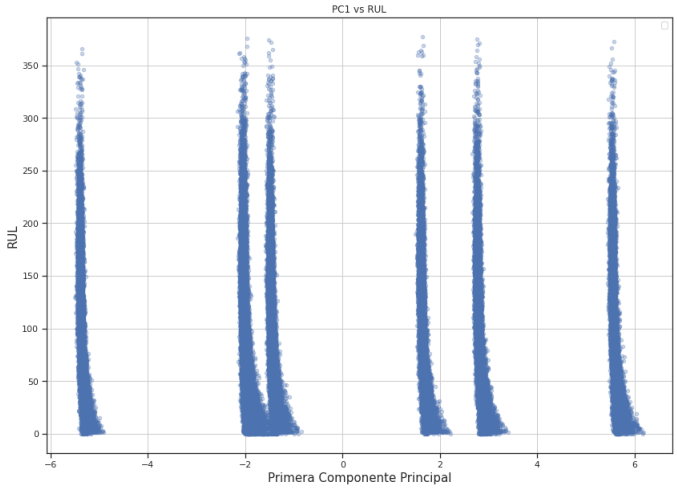



**Figura 4.5:** Gráfico primera componente principal versus RUL del train set FD002
(Fuente: Elaboración propia)

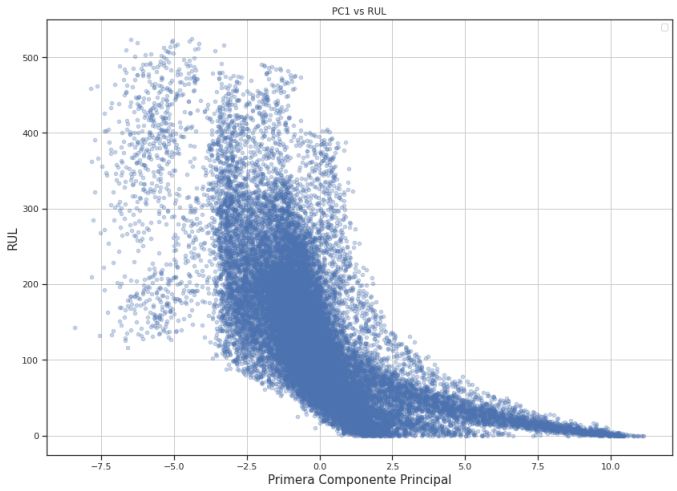



**Figura 4.6:** Gráfico primera componente principal versus RUL del train set FD003
(Fuente: Elaboración propia)

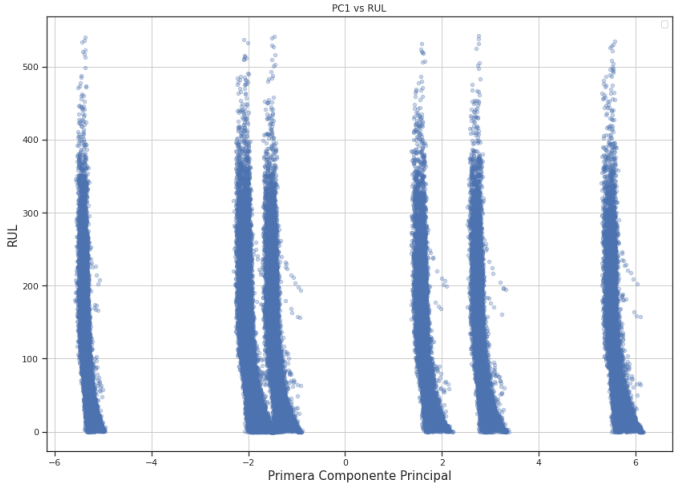



**Figura 4.7:** Gráfico primera componente principal versus RUL del train set FD004
(Fuente: Elaboración propia)

# 5 | Diseño y propuesta

## 5.1. Preprocesamiento Turbofan Engine Degradation Simulation Dataset

### 5.1.1. Selección de features

Del total de features disponibles en el Turbofan Engine Degradation Simulation Dataset, en [15] se dejan fuera los features que no aportan información al entrenamiento del modelo (a diferencia de [16] donde se ocupan todos). Es decir, se descartan los features que se mantienen con un valor constante a lo largo de los 4 datasets. Los features restantes que se consideran son 14 y estos son: sens_2, sens_3, sens_4, sens_7, sens_8, sens_9, sens_11, sens_12, sens_13, sens_14, sens_15, sens_17, sens_20 y sens_21. Se adopta esta selección de características en la propuesta de la solución.

### 5.1.2. Normalización Min-Max

En [19] se destaca la normalización de los datos por dos motivos:

1. Las matrices de pesos se inicializan entre valores [-1,1], por lo que tener features de entrada con diferentes escalas (al evaluar las funciones de activación) generan salidas que no varían entre neuronas de una misma capa (por ejemplo en funciones sigmoide si el valor $z$ es muy grande la salida tiende a 1 y si es pequeño tiende a 0).
2. Dado que existen diferentes escalas en los features, la función $J$ a minimizar también las tendrá. Por lo tanto, la distancia al mínimo y el algoritmo de gradiente descendiente se verán afectados. Normalizando los datos, estas distancias se acortan y el desempeño de la optimización mejora.

La normalización mix-max genera una transformación lineal de un feature dentro de un dataset. Sea $min(x)$ y $max(x)$ el valor máximo y mínimo de un feature, mix-max mapea un valor $x$ cualquiera dentro del feature a un valor $x'$ en un rango $[min(x'), max(x')]$ usando la siguiente fórmula:

$$x' = \frac{x - min(x)}{max(x) - min(x)}(max(x') - min(x')) + min(x') \tag{5.1}$$

En [15] es utilizada la normalización Min-Max en un rango $[min(x'), max(x')] = [-1, 1]$, por lo cual la ecuación anterior queda expresada como:

$$x' = \frac{2(x - min(x))}{max(x) - min(x)} - 1 \tag{5.2}$$

### 5.1.3. Time windows en Turbofan Engine Degradation Simulation Dataset

Se denota como $N_{tw}$ el tamaño de un Time window. En [15] y [16] Los autores usan la técnica de Time windows para crear las muestras de entrada a los modelos de aprendizaje profundo. Este enfoque se adopta porque en una serie de tiempo multivariada la información está contenida tanto en el ciclo actual como en los ciclos anteriores de medición de features de la misma. La estimación del RUL mejora respecto al enfoque de medición de ciclo por ciclo dado que se cuenta con más información histórica.

Por otro lado, si se trabaja con un $N_{tw}$ demasiado grande la predicción carecerá de sentido dado que no será oportuna para tomar decisiones y generar un mantenimiento a tiempo y eficaz. Es necesario tener en cuenta ese equilibrio entre buena estimación del RUL versus $N_{tw}$.

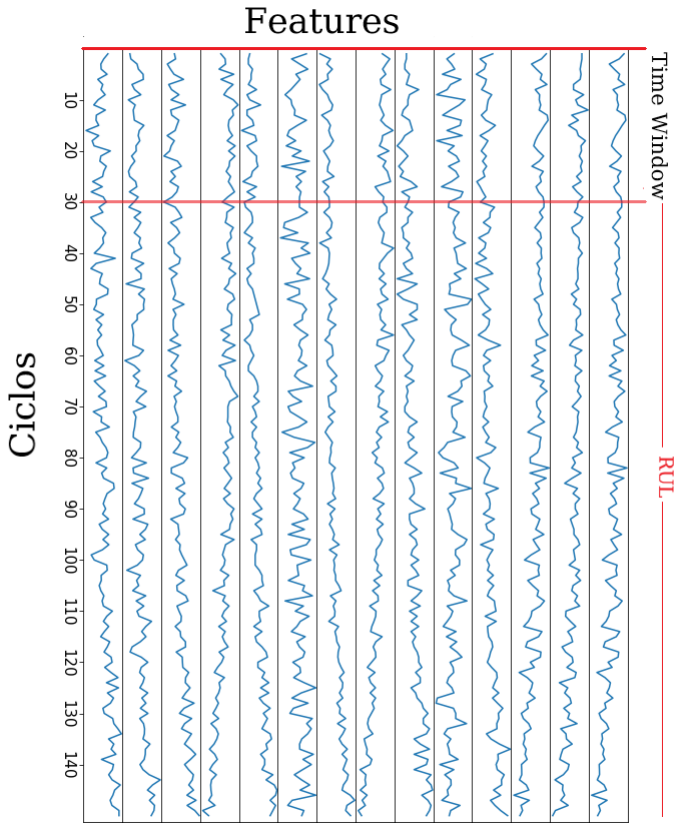



**Figura 5.1:** Time windows en motor del Turbofan Engine Degradation Simulation Dataset con 14 features
(Fuente: Elaboración propia)

En cada test set del Turbofan Engine Degradation Simulation Dataset existe un máximo que puede tener $N_{tw}$, el cual está dado por el motor de cada training y test set con el menor número de ciclos. Teniendo en cuenta lo anterior, el mayor $N_{tw}$ para el dataset FD001 es de 31, $N_{tw} = 21$ para FD002, $N_{tw} = 38$ para FD003 y $N_{tw} = 19$ para FD004.

### 5.1.4. Modificación Target RUL

Se observa en los histogramas del RUL de cada dataset que sus frecuencias pasan de ser constantes a decrecer a partir de RUL = 125 aproximadamente.

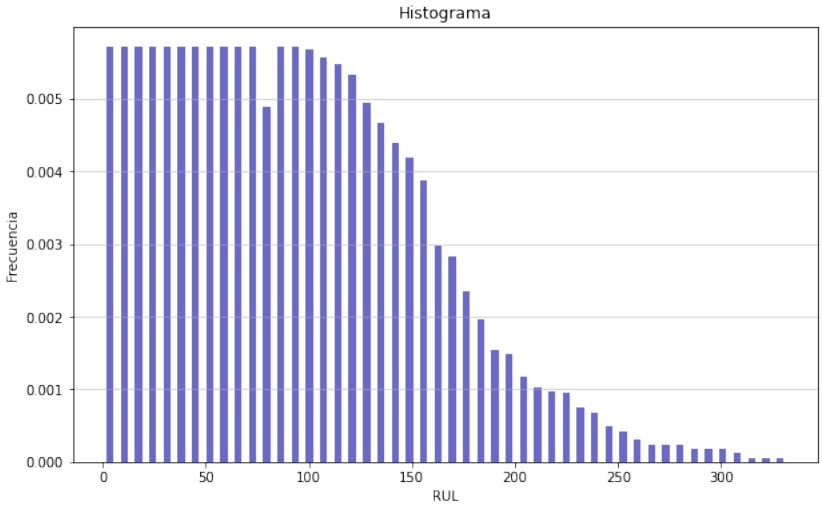



**Figura 5.2:** Histograma RUL dataset FD001
(Fuente: Elaboración propia)

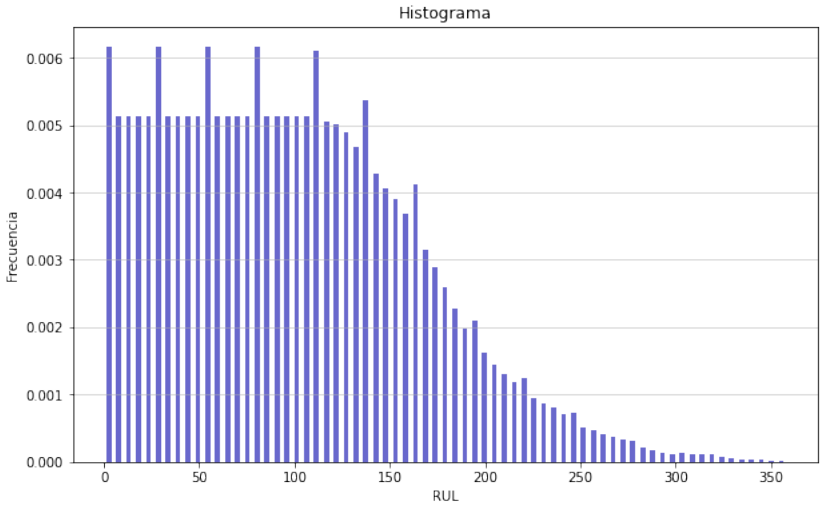



**Figura 5.3:** Histograma RUL dataset FD002
(Fuente: Elaboración propia)

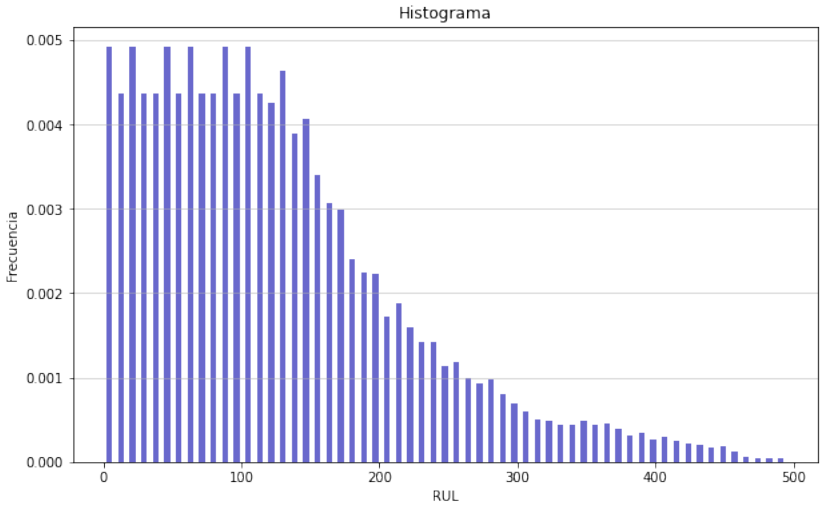



**Figura 5.4:** Histograma RUL dataset FD003
(Fuente: Elaboración propia)

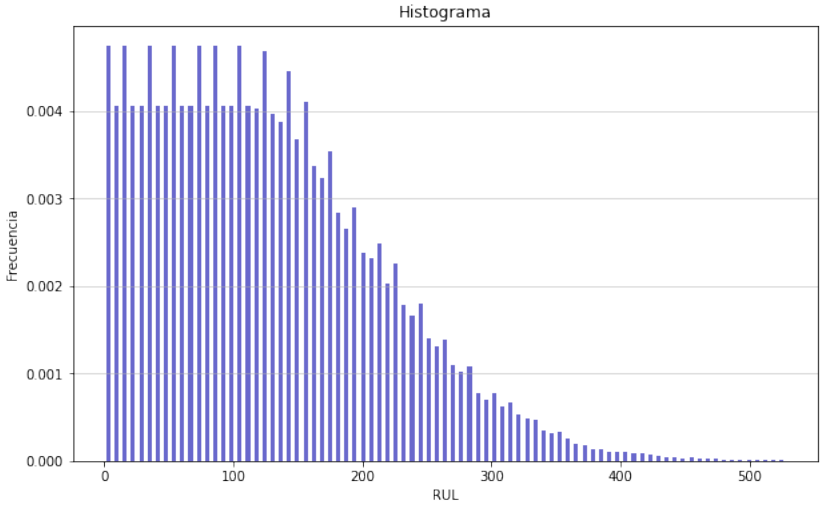



**Figura 5.5:** Histograma RUL dataset FD004
(Fuente: Elaboración propia)

En [15] y [16] todos los RUL > 125 se modifican a RUL = 125 debido a tres razones:

1. Este enfoque genera una mejora en la precisión de la estimación del RUL.
2. Predecir valores de RUL tan altos no traen un beneficio al momento de planificar una mantención ya que se desea saber que tan pronto va a fallar el motor.
3. Evitar el posible subajuste de los modelos generados al entrenarlos a partir de muestras con RUL tan altos.

### 5.1.5. Unión de datasets

Como se vio en la sección 4.2.1 del análisis descriptivo del Turbofan Engine Degradation Simulation Dataset, los datasets FD002 y FD004 tienen un comportamiento parecido entre si asociado a las seis condiciones de operación existentes en ellos. A su vez, los datasets FD001 y FD003 también poseen un comportamiento parecido debido a la única condición de operación existente.

Por lo anterior se realiza una unión de datasets con el fin de mejorar el desempeño de los modelos a entrenar, ya que al contar con una mayor cantidad de datos estos modelos pueden aprender nuevas representaciones lo cual les permite generalizar de mejor manera a nuevas muestras que no han visto nunca. Se generan dos datasets: FD001+FD003 y FD002+FD004. Con esta unión los datasets quedan conformados de la siguiente forma:

**Tabla 5.1:** Detalle Turbofan Engine Degradation Simulation Dataset modificado con unión de datasets

| Dataset | N° Motores train set | N° Motores test set | Cond. Operación | Modos de falla |
|---|---|---|---|---|
| FD001 + FD003 | 200 | 200 | 1 | 2 |
| FD002 + FD004 | 508 | 508 | 6 | 2 |

## 5.2. Configuración de hiperparámetros

Se da enfásis a la configuración de los siguientes hiperparámetros:

- **N° Epochs:** La cantidad de epochs usados en el entrenamiento de un modelo de aprendizaje profundo afecta directamente a la cantidad de veces que se ejecuta la optimización de las matrices de pesos, forward y back-propagation. Entre más epochs se configuren en el entrenamiento del modelo, el train error disminuye. Hay que tener en cuenta que si se escoge una cantidad muy alta de epochs se corre el riesgo de sobreajustar el modelo. En [15] se escoge una cantidad de 250 epochs.
- **Batch Size:** En [20] se postula que existe una compensación entre el batch size versus el tiempo de entrenamiento del modelo de aprendizaje profundo: entre más grande es el batch size las predicciones del modelo mejoran y este es capaz de generalizar con muestras nuevas(test set), pero al mismo tiempo el costo computacional crece. En [15] se escoge un batch size de 512.

## 5.3. Dropout

El Dropout [21] es una técnica usada en [15] la cual en el proceso de entrenamiento del modelo de aprendizaje profundo desactiva neuronas en la capa oculta deseada de la red neuronal con una probabilidad $p$ definida previamente. En el proceso de evaluación, las neuronas deshabilitadas se activan nuevamente. Esta técnica reduce el sobreajuste del modelo de aprendizaje profundo.

## 5.4. Optimizadores

En [15] se ocupa Adam como optimizador. A su vez, en [16] es usado SGD. Por el excelente desempeño que logra Adam respecto a otros optimizadores vistos en la sección 2.6, se escoge este optimizador para el desarrollo de la solución.

## 5.5. Modificaciones a las CNNs del estado del arte

En [6] se destaca que entre más representaciones complejas de los datos pueda capturar un modelo de aprendizaje automático, este modelo tendrá un menor train error (mayor capacidad). A lo anterior se debe tener en cuenta que si el modelo es muy complejo es más probable que ocurra sobreajuste. El agregar más capas (Convolucionales, Fully-Connected) aumenta la capacidad de las arquitecturas de aprendizaje profundo para encontrar representaciones más complejas en los datos y a su vez el uso de Dropout permite atenuar el sobreajuste que pueda ocurrir al agregarlas.

Se escogen las arquitecturas de [15] y [16] para ser adaptadas. Se espera ver el efecto de las modificaciones en el desempeño de estas. A la arquitectura de Li, Ding y Sun [15] se agregan dos capas más de convolución idénticas a las cuatro primeras capas convolucionales con 10 filtros de 10x1 cada una, aplicando a su vez la técnica de same-padding. Además, se adicionan más neuronas a la capa fully-connected final de 100 a 256 neuronas manteniendo la función de activación *tanh*. Se aplica en esta capa la técnica de Dropout con $p = 0{,}5$.

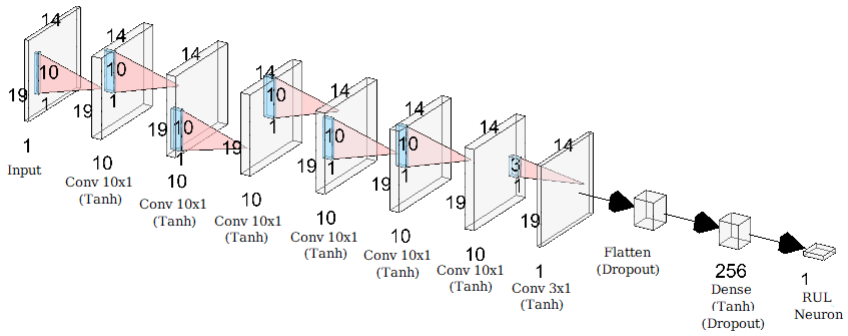



**Figura 5.6:** Arquitectura propuesta basada en Li, Ding y Sun
(Fuente: Elaboración propia)

Por otro lado, a la arquitectura de Babu, Zhao y Li [16] se agrega una capa fully-connected de 1024 neuronas con función de activación *sigmoide*. También es aplicada la técnica de Dropout con $p = 0{,}5$. Por último, en las capas convolucionales de la arquitectura se aplica la técnica de same-padding.

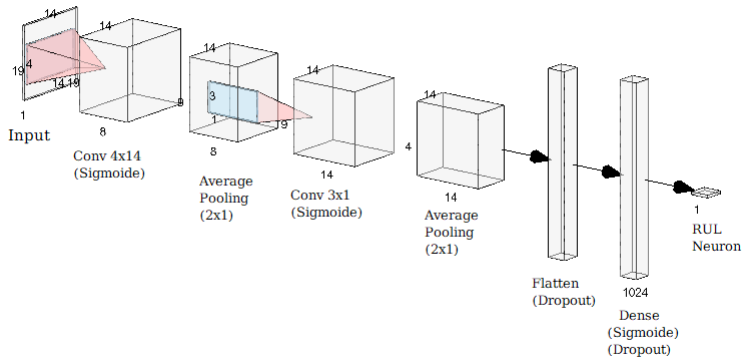



**Figura 5.7:** Arquitectura propuesta basada en Babu, Zhao y Li
(Fuente: Elaboración propia)

## 5.6. Métricas de evaluación

Para obtener una métrica de evaluación del modelo en [15] y [16] se utiliza el Root Mean Square Error:

$$\textbf{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} h_i^2} \tag{5.3}$$

donde $N$ es el total de motores en cada test set y :

$$h_i = RUL \text{ estimado} - RUL \text{ verdadero} \tag{5.4}$$

es el error entre el RUL verdadero del motor $i$ del test set y su predicción de RUL usando el modelo de aprendizaje profundo. A su vez, se ocupa esta métrica para calcular el RMSE por tramos:

- Tramo 1 (early): Motores del test set con RUL verdadero $\in [1, 40]$.
- Tramo 2 (mid): Motores del test set con RUL verdadero $\in [41, 80]$.
- Tramo 3 (late): Motores del test set con RUL verdadero $\in [81, 120]$.

Se desea ver que tan precisa es la predicción de los modelos ante diferentes intervalos de RUL. El tramo 1 es el más importante debido a su cercanía con la falla del motor de avión. En un caso real la planificación de mantenimiento predictivo se hará confiando en la predicción de los modelos en este tramo.

# 6 | Resultados

## 6.1. Configuraciones de entrenamiento

Para generar el train set de los modelos se utilizan:

- 14 features mencionados en la sección 5.1.1
- Normalización min-max de los datos entre el rango [-1,1]
- Time windows con tamaño $N_{tw} = 10$ y $N_{tw} = 19$
- Se modifica todo RUL > 120 a un RUL = 120 esperando que conlleve una mejora en la métricas de evaluación de los modelos de acuerdo a lo mencionado en la sección 5.1.4.
- Tras el análisis de la sección 5.1.5 se fusionan los datasets FD001+FD003 y FD002+FD004. Con esta modificación el train set de cada modelo crece considerablemente.
- Se escogen dos valores de epochs: $E = 250$ y $E = 500$.
- Dos tamaños de batch size son usados: $B_{size} = 512$ y $B_{size} = 1024$.
- Como se mencionó anteriormente en la sección 5.4, el optimizador utilizado es Adam.
- Las arquitecturas de aprendizaje profundo supervisado usadas son las arquitecturas modificadas mencionadas en la sección 5.5.
- El 10 % del train set se usa como validation set (dev set).

Teniendo en cuenta lo anterior, se generan 32 modelos usando ambas arquitecturas (16 modelos cada una). En cada arquitectura se generan modelos distintos para los dataset FD001+FD003 y FD002+FD004 usando las siguientes configuraciones:

**Tabla 6.1:** Detalle configuraciones entrenamiento para arquitecturas modificadas con dataset FD001+FD003.

| $N_{tw}$ × Features | Entradas train set | Entradas dev set | $B_{size}$ | $E$ |
|---|---|---|---|---|
| 19×14 | 37575 | 4176 | 512 | 250 |
| 19×14 | 37575 | 4176 | 1024 | 250 |
| 19×14 | 37575 | 4176 | 512 | 500 |
| 19×14 | 37575 | 4176 | 1024 | 500 |
| 10×14 | 39195 | 4356 | 512 | 250 |
| 10×14 | 39195 | 4356 | 1024 | 250 |
| 10×14 | 39195 | 4356 | 512 | 500 |
| 10×14 | 39195 | 4356 | 1024 | 500 |

**Tabla 6.2:** Detalle configuraciones entrenamiento para arquitecturas modificadas con dataset FD002+FD004.

| $N_{tw}$ × Features | Entradas train set | Entradas dev set | $B_{size}$ | $E$ |
|---|---|---|---|---|
| 19×14 | 95261 | 10585 | 512 | 250 |
| 19×14 | 95261 | 10585 | 1024 | 250 |
| 19×14 | 95261 | 10585 | 512 | 500 |
| 19×14 | 95261 | 10585 | 1024 | 500 |
| 10×14 | 99384 | 11043 | 512 | 250 |
| 10×14 | 99384 | 11043 | 1024 | 250 |
| 10×14 | 99384 | 11043 | 512 | 500 |
| 10×14 | 99384 | 11043 | 1024 | 500 |

Es utilizado Keras para definir las arquitecturas, entrenar y evaluar los modelos generados. Además son usados los entornos de desarrollo de Google Colab y JupyterLab, los cuales proveen Python 3.6 y distintas librerías de Data Science tales como Scikit-Learn, Pandas, numpy y matplotlib. Para acelerar el entrenamiento de los modelos se utiliza la GPU proporcionada por Google Colab y una GPU NVIDIA GTX 1660Ti que admite CUDA y es compatible con Keras.

## 6.2. Resultados test set

Para obtener los resultados presentados a continuación se generan 5 experimentos por cada modelo con el fin de atenuar la aleatoriedad producida al entrenarlos. Así es obtenida una medición de RMSE, RMSE early en el tramo early, RMSE mid en el tramo mid y RMSE late en el tramo late con una media y desviación estándar. El nombre de cada modelo tiene el formato AA_$N_{tw}$_$B_{size}$_$E$_X_Y donde:

- AA = Li señala que el modelo respectivo es entrenado usando la arquitectura propuesta basada en Li, Ding. Sun. Por otro lado, si AA = Ba el modelo respectivo es entrenado usando la arquitectura propuesta basada en Babu, Zhao y Li.
- $N_{tw}$ corresponde al tamaño de Time windows utilizado para crear las muestras de entrada al modelo respectivo.
- $B_{size}$ corresponde al tamaño de batch size utilizado para entrenar el modelo respectivo.
- $E$ corresponde al número de epochs utilizado para entrenar el modelo respectivo.
- X_Y = 1_3 señala que el modelo respectivo es entrenado usando el dataset FD001+FD003. Por otro lado, si X_Y = 2_4 el modelo respectivo es entrenado usando el dataset FD002+FD004.

Los resultados de las métricas de evaluación de los modelos usando la arquitectura propuesta basada en Li, Ding y Sun son:

**Tabla 6.3:** Resultados RMSE, RMSE early, RMSE mid y RMSE late de modelos entrenados utilizando arquitectura propuesta basada en Li, Ding y Sun y dataset FD001+FD003.

| Nombre modelo | RMSE | RMSE early | RMSE mid | RMSE late | $N_{tw}$ | $B_{size}$ | $E$ |
|---|---|---|---|---|---|---|---|
| Li_19_512_250_1_3 | 14,18 ± 0,36 | **6,88 ± 1,27** | 19,5 ± 0,99 | 13,93 ± 0,62 | 19 | 512 | 250 |
| Li_19_1024_250_1_3 | **13,88 ± 0,28** | 6,36 ± 0,51 | 19 ± 0,55 | **13,81 ± 0,29** | 19 | 1024 | 250 |
| Li_19_512_500_1_3 | **14,56 ± 0,75** | **6,14 ± 0,51** | **19,9 ± 1,28** | **14,6 ± 0.73** | 19 | 512 | 500 |
| Li_19_1024_500_1_3 | 13,91 ± 0,39 | 6,39 ± 0,57 | **18,97 ± 0.91** | 13,88 ± 0,21 | 19 | 1024 | 500 |
| Li_10_512_250_1_3 | 17,23 ± 0,34 | **14,28 ± 0,41** | 25,03 ± 0,74 | 13,82 ± 0,25 | 10 | 512 | 250 |
| Li_10_1024_250_1_3 | **17 ± 0,33** | 13,7 ± 0,66 | **24,74 ± 0,62** | **13,8 ± 0,15** | 10 | 1024 | 250 |
| Li_10_512_500_1_3 | **17,5 ± 0,37** | 14,18 ± 1,06 | **25,38 ± 0,49** | **14,24 ± 0,14** | 10 | 512 | 500 |
| Li_10_1024_500_1_3 | 17,2 ± 0,44 | **13,63 ± 0,62** | 25,02 ± 1,23 | 14,07 ± 0,26 | 10 | 1024 | 500 |

**Tabla 6.4:** Resultados RMSE, RMSE early, RMSE mid y RMSE late de modelos entrenados utilizando arquitectura propuesta basada en Li, Ding y Sun y dataset FD002+FD004.

| Nombre modelo | RMSE | RMSE early | RMSE mid | RMSE late | $N_{tw}$ | $B_{size}$ | $E$ |
|---|---|---|---|---|---|---|---|
| Li_19_512_250_2_4 | 22,51 ± 0,82 | 23,09 ± 2,71 | **30,4 ± 2,42** | 18,13 ± 1,93 | 19 | 512 | 250 |
| Li_19_1024_250_2_4 | **23,97 ± 1,47** | **25,36 ± 6,03** | **28,17 ± 4,37** | **20,63 ± 3,77** | 19 | 1024 | 250 |
| Li_19_512_500_2_4 | 23,06 ± 1,44 | **22,33 ± 6,43** | 29,34 ± 3,02 | 20 ± 3,29 | 19 | 512 | 500 |
| Li_19_1024_500_2_4 | **21,84 ± 0,69** | 22,43 ± 2,86 | 29,76 ± 3,07 | **17,32 ± 2,2** | 19 | 1024 | 500 |
| Li_10_512_250_2_4 | 20,46 ± 1,05 | **20,46 ± 3,14** | **29,54 ± 3,3** | **15,45 ± 2,11** | 10 | 512 | 250 |
| Li_10_1024_250_2_4 | **21,02 ± 1,18** | **17,48 ± 1,83** | **26,06 ± 1,24** | **20,51 ± 2,95** | 10 | 1024 | 250 |
| Li_10_512_500_2_4 | **20,06 ± 0,66** | 17,96 ± 2,86 | 27,36 ± 2,21 | 17,56 ± 2,12 | 10 | 512 | 500 |
| Li_10_1024_500_2_4 | 20,28 ± 0,32 | 17,79 ± 1,9 | 26,52 ± 2,06 | 18,61 ± 2,35 | 10 | 1024 | 500 |

A su vez, los resultados de las métricas de evaluación de los modelos usando la arquitectura propuesta basada en Babu, Zhao y Li son:

**Tabla 6.5:** Resultados RMSE, RMSE early, RMSE mid y RMSE late de modelos entrenados utilizando arquitectura propuesta basada en Babu, Zhao y Li y dataset FD001+FD003.

| Nombre modelo | RMSE | RMSE early | RMSE mid | RMSE late | $N_{tw}$ | $B_{size}$ | $E$ |
|---|---|---|---|---|---|---|---|
| Ba_19_512_250_1_3 | 15,69 ± 0,41 | 12,19 ± 1,26 | 21,96 ± 0,7 | 13,62 ± 0,28 | 19 | 512 | 250 |
| Ba_19_1024_250_1_3 | **16,05 ± 0,3** | **13,29 ± 0,61** | **22,58 ± 0,59** | **13,48 ± 0,22** | 19 | 1024 | 250 |
| Ba_19_512_500_1_3 | 15,83 ± 0,39 | 11,18 ± 0,74 | 22,19 ± 1,25 | **14,17 ± 0,44** | 19 | 512 | 500 |
| Ba_19_1024_500_1_3 | **15,37 ± 0,27** | **10,85 ± 0,35** | **21,88 ± 0,72** | 13,55 ± 0,23 | 19 | 1024 | 500 |
| Ba_10_512_250_1_3 | **17,75 ± 0,36** | 16,01 ± 0,98 | **25,13 ± 0,73** | **14,08 ± 0,17** | 10 | 512 | 250 |
| Ba_10_1024_250_1_3 | **18,12 ± 0,36** | **16,14 ± 0,69** | **25,97 ± 0,68** | 14,24 ± 0,17 | 10 | 1024 | 250 |
| Ba_10_512_500_1_3 | 17,81 ± 0,56 | 15,6 ± 0,92 | 25,37 ± 1,35 | 14,24 ± 0,29 | 10 | 512 | 500 |
| Ba_10_1024_500_1_3 | 17,82 ± 0,26 | **15,31 ± 0,38** | 25,56 ± 0,32 | **14,31 ± 0,26** | 10 | 1024 | 500 |

**Tabla 6.6:** Resultados RMSE, RMSE early, RMSE mid y RMSE late de modelos entrenados utilizando arquitectura propuesta basada en Babu, Zhao y Li y dataset FD002+FD004.

| Nombre modelo | RMSE | RMSE early | RMSE mid | RMSE late | $N_{tw}$ | $B_{size}$ | $E$ |
|---|---|---|---|---|---|---|---|
| Ba_19_512_250_2_4 | **18,89 ± 0,74** | 16,86 ± 2,12 | 26,39 ± 1,86 | **16,32 ± 0,78** | 19 | 512 | 250 |
| Ba_19_1024_250_2_4 | **19,3 ± 0,5** | **17,69 ± 0,86** | **26,76 ± 0,55** | 16,6 ± 0,79 | 19 | 1024 | 250 |
| Ba_19_512_500_2_4 | 18,895 ± 0,29 | 16,14 ± 1,45 | 25,89 ± 1,67 | 16,99 ± 1,34 | 19 | 512 | 500 |
| Ba_19_1024_500_2_4 | 18,899 ± 0,36 | **15,68 ± 1,12** | **25,56 ± 1** | **17,48 ± 0,64** | 19 | 1024 | 500 |
| Ba_10_512_250_2_4 | 18,4 ± 0,26 | **14,42 ± 1,05** | 25,51 ± 0,73 | **17,09 ± 0,69** | 10 | 512 | 250 |
| Ba_10_1024_250_2_4 | **19,25 ± 0,89** | **16,67 ± 0,83** | **27,34 ± 1,84** | 16,7 ± 0,77 | 10 | 1024 | 250 |
| Ba_10_512_500_2_4 | **18,05 ± 0,32** | 15,06 ± 1,23 | **25,43 ± 0,77** | 16,09 ± 0,35 | 10 | 512 | 500 |
| Ba_10_1024_500_2_4 | 18,33 ± 0,23 | 15,54 ± 1,14 | 26,47 ± 0,45 | **15,82 ± 0,41** | 10 | 1024 | 500 |

## 6.3. Gráficos

A continuación se adjuntan gráficos de dispersión de RUL verdadero versus RUL estimado de cada motor del test set (de su respectivo dataset) usando cada modelo nombrado anteriormente en las tablas 6.3 a la 6.6. Para generar estos gráficos se ordenan los motores del test set a partir de su RUL verdadero en forma decreciente, así luego es posible graficar sus valores de RUL verdadero (puntos rojos) y RUL estimado (puntos azules) generados por la predicción de los respectivos modelos entrenados anteriormente. Adicionalmente, se resalta cada tramo con un color distintivo que alude a la condición de degradación actual (RUL verdadero) del motor: tramo late de color verde, tramo mid de color amarillo y tramo early (los más cercanos a RUL = 0 y por ende a la falla) de color rojo.

Por otro lado, se generan gráficos de training y validation error versus número de epochs de cada modelo nombrado anteriormente en las tablas 6.3 a la 6.6. Se utiliza $Log(Error)$ en base 10 para facilitar la lectura del valor del eje Y.

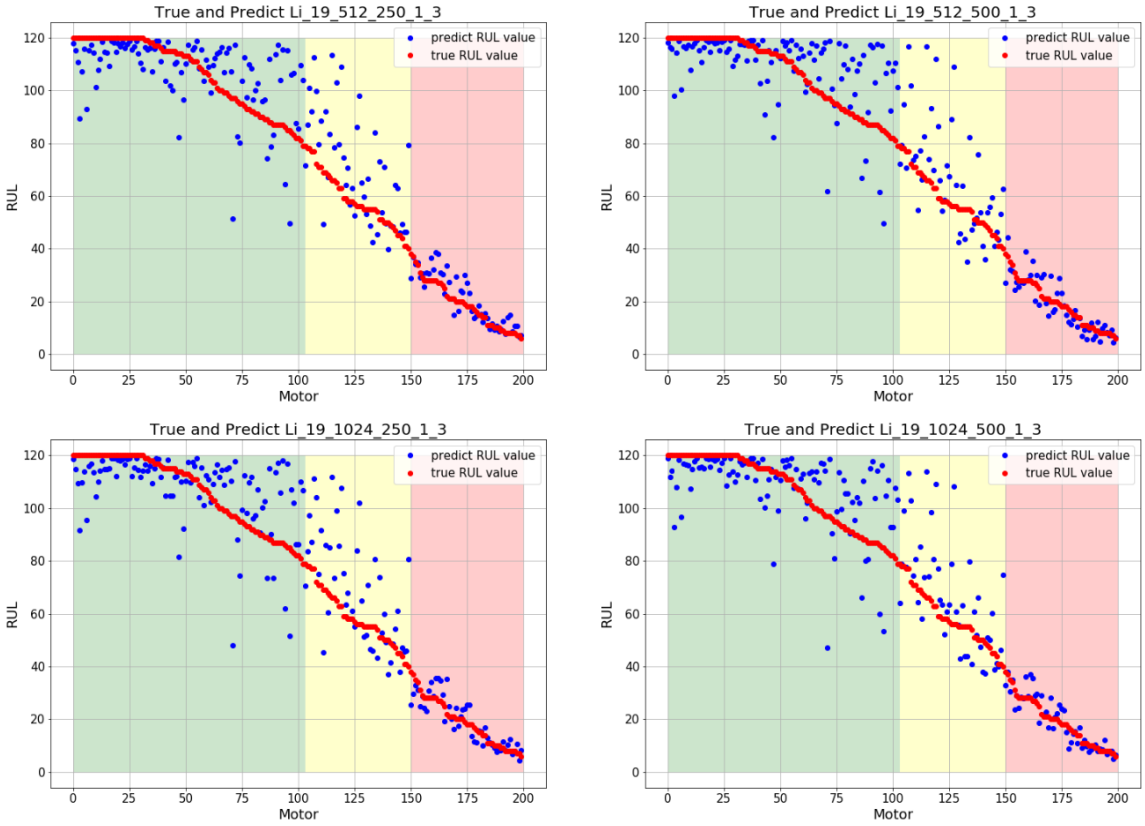



**Figura 6.1:** Gráficos de dispersión RUL verdadero y estimado de cada motor del test set usando modelos Li_19_512_250_1_3, Li_19_512_500_1_3, Li_19_1024_250_1_3 y Li_19_1024_500_1_3

(Fuente: Elaboración propia)

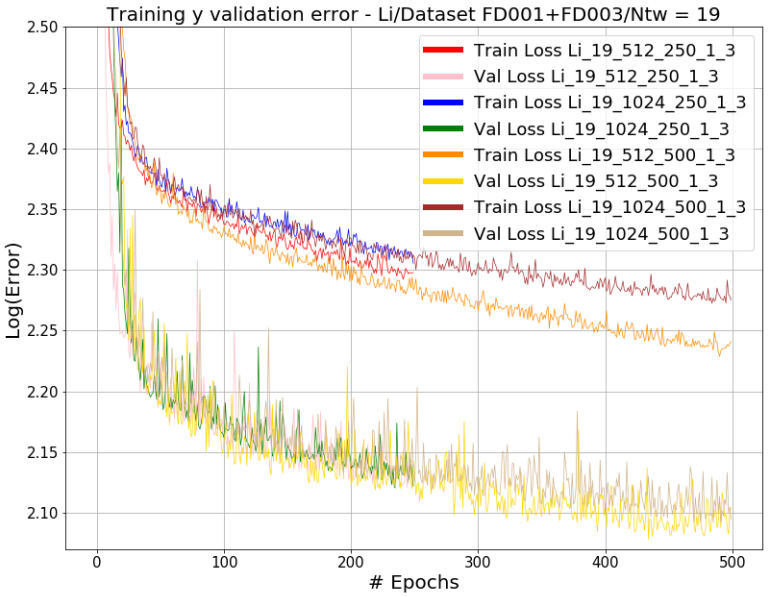



**Figura 6.2:** Gráfico training y validation error vs número de epochs modelos Li_19_512_250_1_3, Li_19_512_500_1_3, Li_19_1024_250_1_3 y Li_19_1024_500_1_3

(Fuente: Elaboración propia)

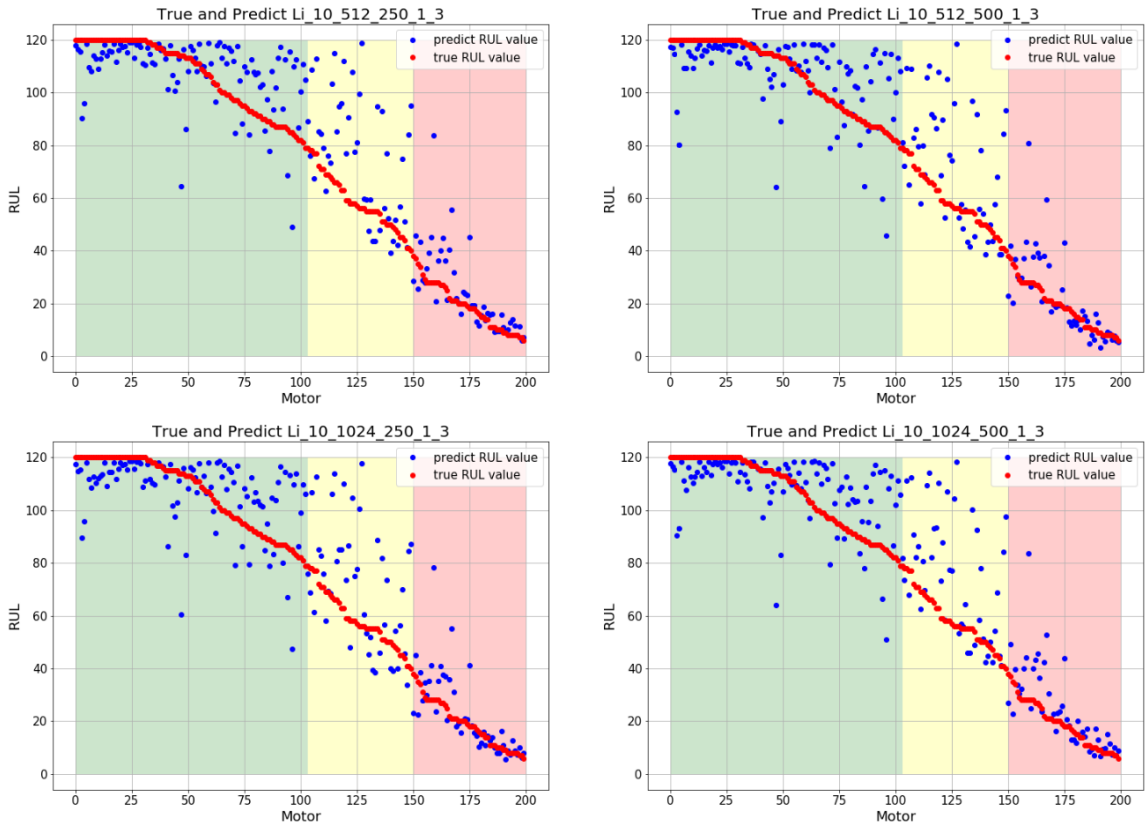



**Figura 6.3:** Gráficos de dispersión RUL verdadero y estimado de cada motor del test set usando modelos Li_10_512_250_1_3, Li_10_512_500_1_3, Li_10_1024_250_1_3 y Li_10_1024_500_1_3

(Fuente: Elaboración propia)

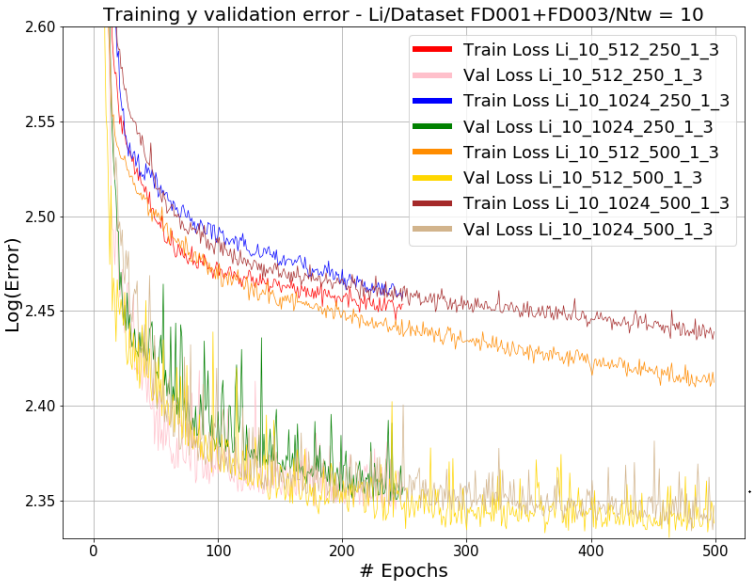



**Figura 6.4:** Gráfico training y validation error vs número de epochs modelos Li_10_512_250_1_3, Li_10_512_500_1_3, Li_10_1024_250_1_3 y Li_10_1024_500_1_3

(Fuente: Elaboración propia)

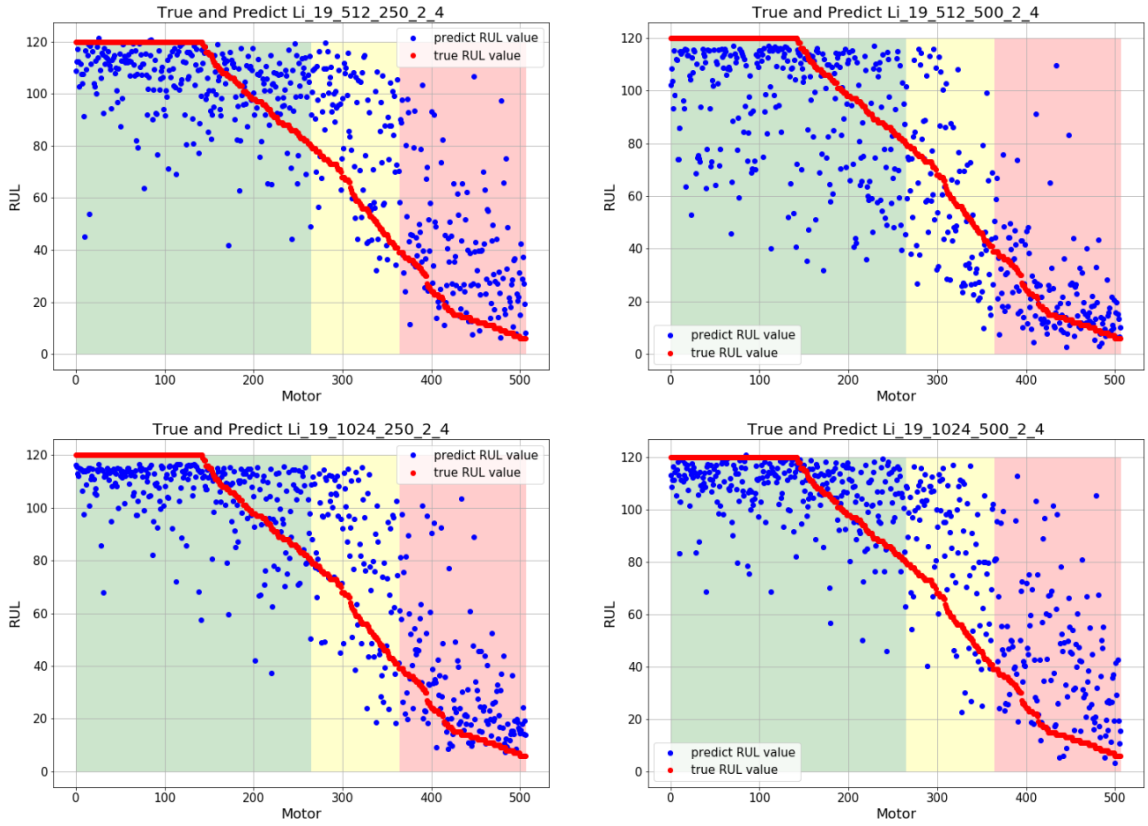



**Figura 6.5:** Gráficos de dispersión RUL verdadero y estimado de cada motor del test set usando modelos Li_19_512_250_2_4, Li_19_512_500_2_4, Li_19_1024_250_2_4 y Li_19_1024_500_2_4

(Fuente: Elaboración propia)

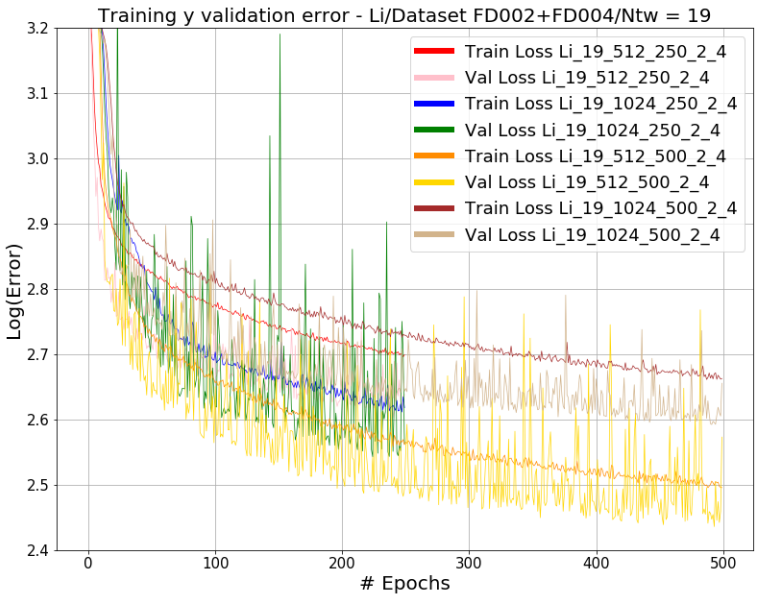



**Figura 6.6:** Gráfico training y validation error vs número de epochs modelos Li_19_512_250_2_4, Li_19_512_500_2_4, Li_19_1024_250_2_4 y Li_19_1024_500_2_4

(Fuente: Elaboración propia)

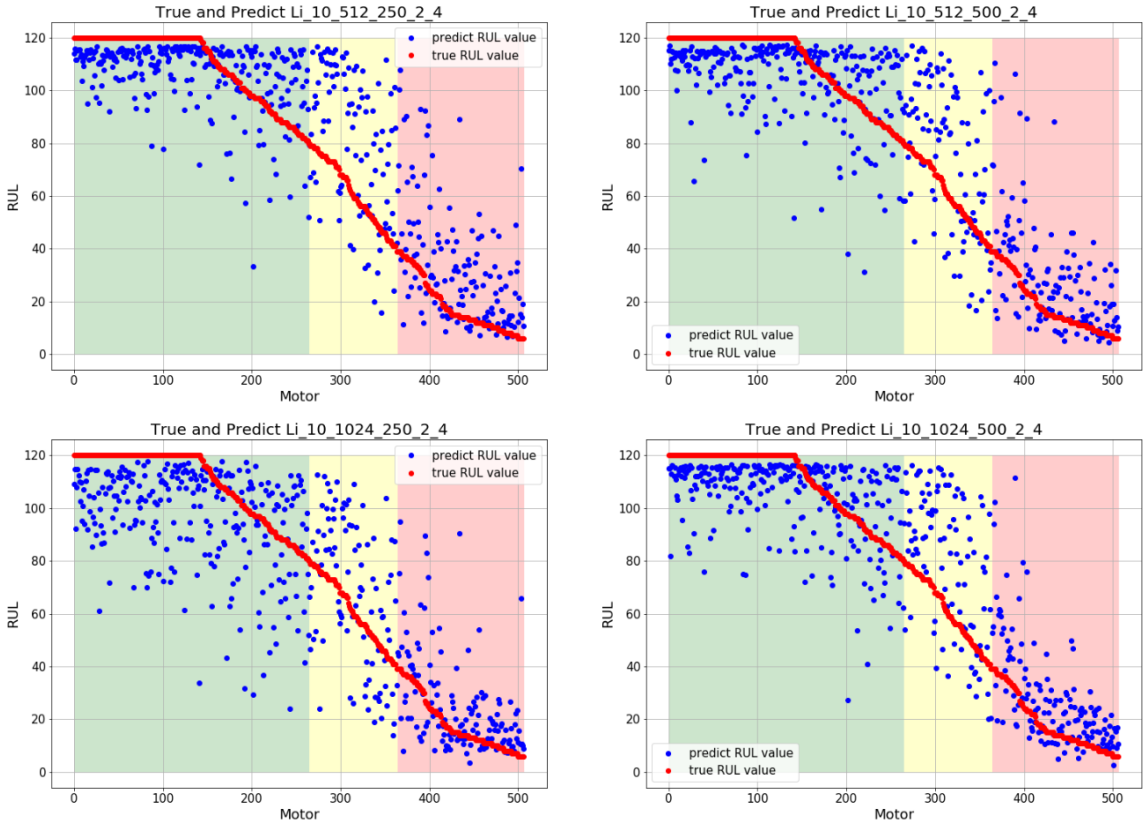



**Figura 6.7:** Gráficos de dispersión RUL verdadero y estimado de cada motor del test set usando modelos Li_10_512_250_2_4, Li_10_512_500_2_4, Li_10_1024_250_2_4 y Li_10_1024_500_2_4

(Fuente: Elaboración propia)

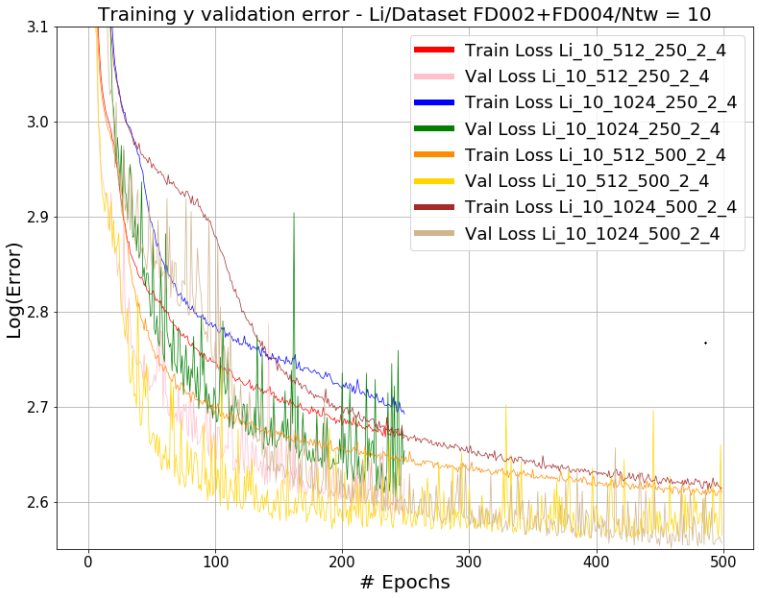



**Figura 6.8:** Gráfico training y validation error vs número de epochs modelos Li_10_512_250_2_4, Li_10_512_500_2_4, Li_10_1024_250_2_4 y Li_10_1024_500_2_4

(Fuente: Elaboración propia)

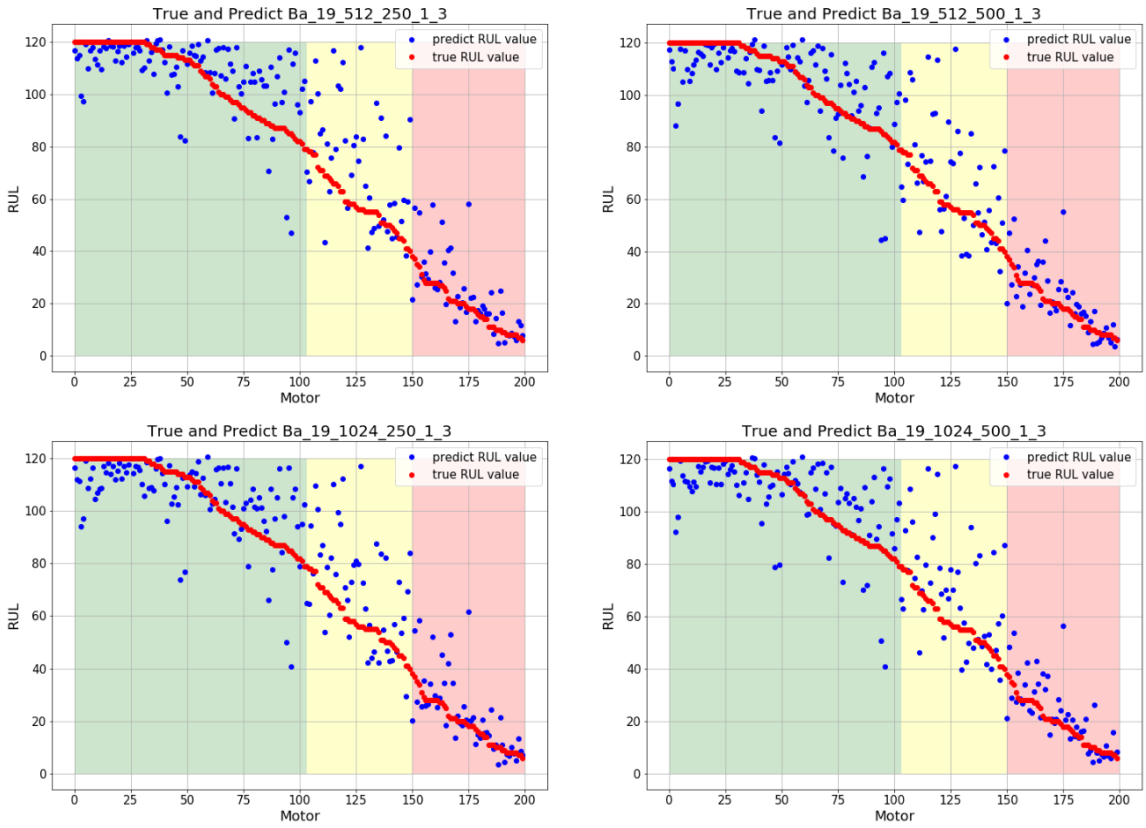



**Figura 6.9:** Gráficos de dispersión RUL verdadero y estimado de cada motor del test set usando modelos Ba_19_512_250_1_3, Ba_19_512_500_1_3, Ba_19_1024_250_1_3 y Ba_19_1024_500_1_3

(Fuente: Elaboración propia)

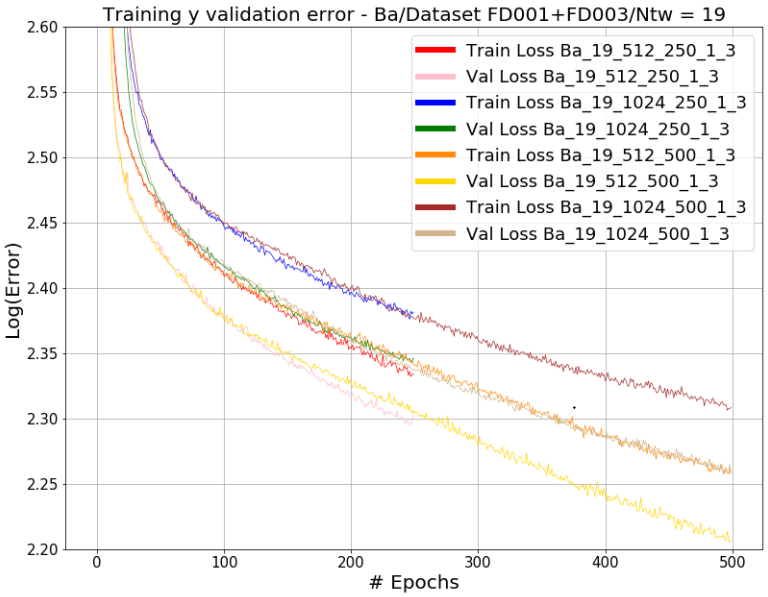



**Figura 6.10:** Gráfico training y validation error vs número de epochs modelos Ba_19_512_250_1_3, Ba_19_512_500_1_3, Ba_19_1024_250_1_3 y Ba_19_1024_500_1_3

(Fuente: Elaboración propia)

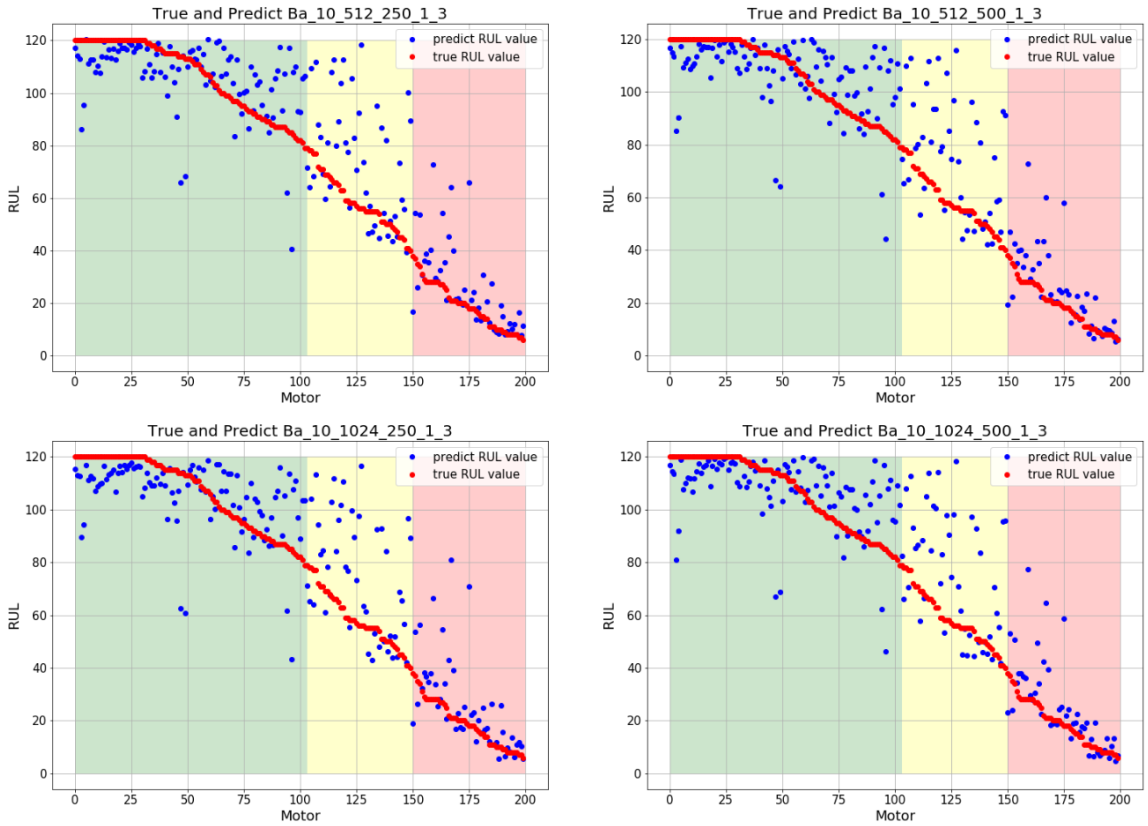



**Figura 6.11:** Gráficos de dispersión RUL verdadero y estimado de cada motor del test set usando modelos Ba_10_512_250_1_3, Ba_10_512_500_1_3, Ba_10_1024_250_1_3 y Ba_10_1024_500_1_3

(Fuente: Elaboración propia)

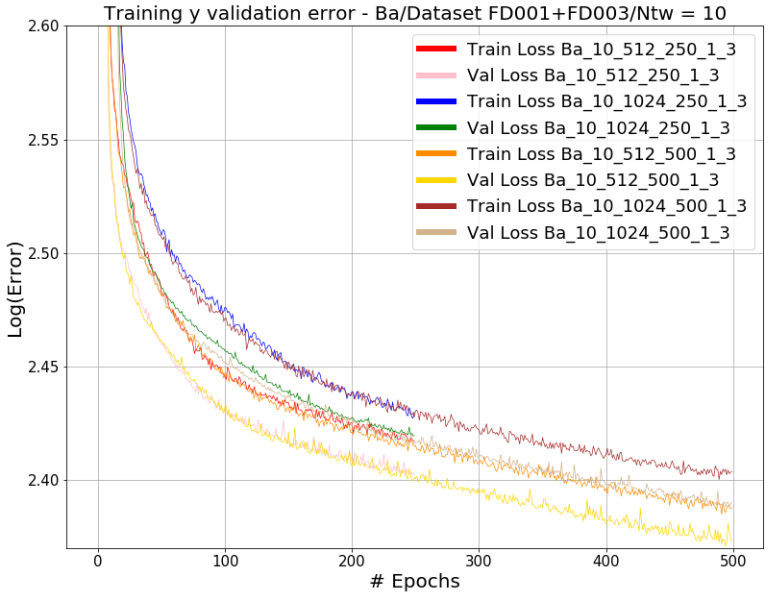



**Figura 6.12:** Gráfico training y validation error vs número de epochs modelos Ba_10_512_250_1_3, Ba_10_512_500_1_3, Ba_10_1024_250_1_3 y Ba_10_1024_500_1_3

(Fuente: Elaboración propia)

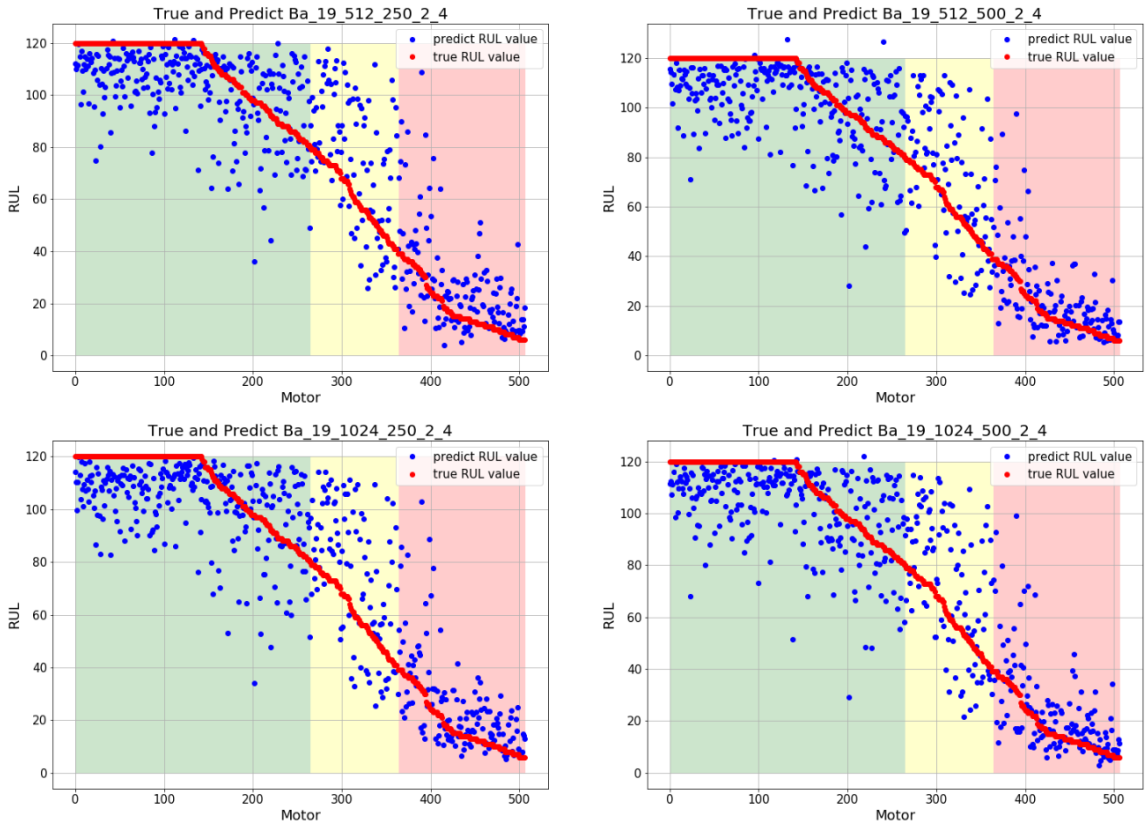



**Figura 6.13:** Gráficos de dispersión RUL verdadero y estimado de cada motor del test set usando modelos Ba_19_512_250_2_4, Ba_19_512_500_2_4, Ba_19_1024_250_2_4 y Ba_19_1024_500_2_4

(Fuente: Elaboración propia)

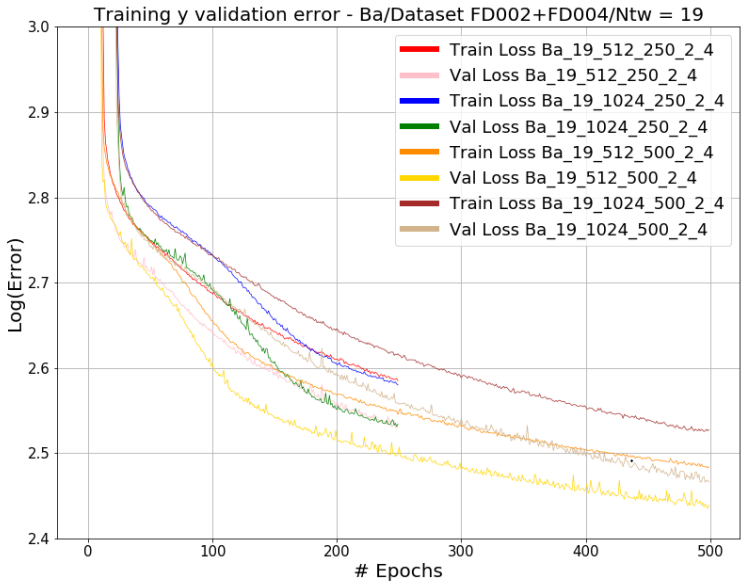



**Figura 6.14:** Gráfico training y validation error vs número de epochs modelos Ba_19_512_250_2_4, Ba_19_512_500_2_4, Ba_19_1024_250_2_4 y Ba_19_1024_500_2_4

(Fuente: Elaboración propia)

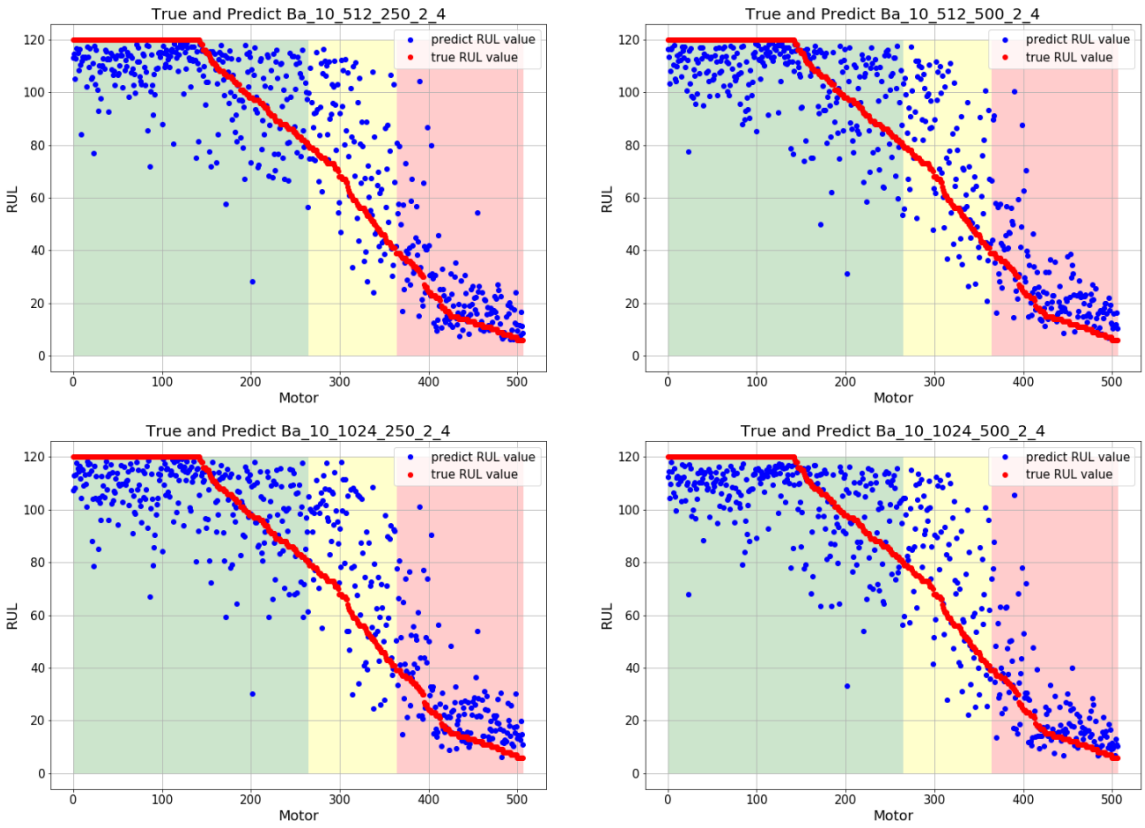



**Figura 6.15:** Gráficos de dispersión RUL verdadero y estimado de cada motor del test set usando modelos Ba_10_512_250_2_4, Ba_10_512_500_2_4, Ba_10_1024_250_2_4 y Ba_10_1024_500_2_4

(Fuente: Elaboración propia)

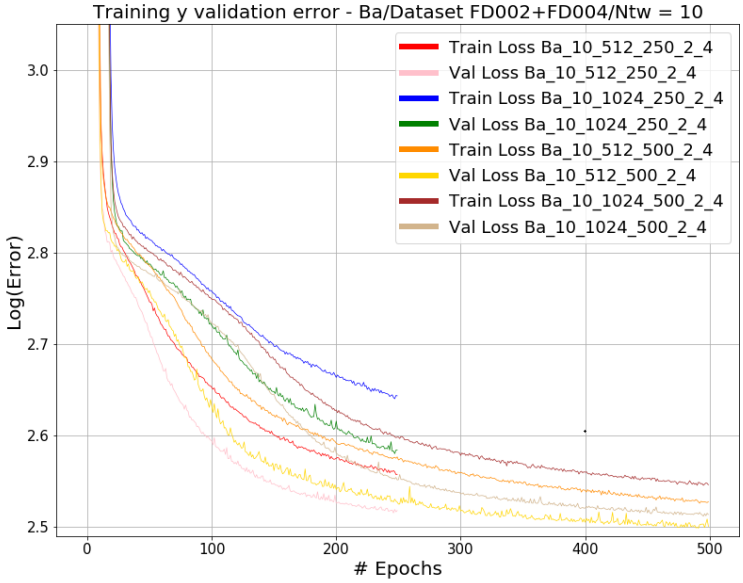



**Figura 6.16:** Gráfico training y validation error vs número de epochs modelos Ba_10_512_250_2_4, Ba_10_512_500_2_4, Ba_10_1024_250_2_4 y Ba_10_1024_500_2_4

(Fuente: Elaboración propia)

# 7 | Discusión de resultados

Analizando los modelos entrenados con el dataset FD001+FD003 y con $N_{tw} = 19$ se puede observar que el modelo **Li_19_1024_250_1_3** tiene el menor valor de RMSE y un RMSE mid muy cercano al modelo con la mejor métrica de este tipo: el modelo **Li_19_1024_500_1_3**. En esta misma línea, el modelo **Li_19_512_500_1_3** obtiene el mayor valor de RMSE late y a su vez el menor RMSE early de todos los modelos con los criterios mencionados anteriormente. Cabe destacar además que el modelo **Ba_19_1024_250_1_3** obtiene los mayores valores de RMSE, RMSE mid pero a su vez el menor valor de RMSE late.

Con $N_{tw} = 10$ se observa que el modelo **Li_10_1024_250_1_3** posee el menor valor de RMSE, RMSE mid y RMSE late además de un valor muy cercano de RMSE early al modelo que obtiene el menor valor de esta métrica: el modelo **Li_10_1024_500_1_3**. Dado lo anterior se observa lo siguiente:

1. Un mayor tamaño de $N_{tw}$ trae una mejora en la predicción de los modelos entrenados con el dataset FD001+FD003, independiente de la arquitectura de los mismos.
2. Ante el mismo dataset FD001+FD003 y los mismos hiperparámetros tales como cantidad de epochs, batch size y $N_{tw}$ la arquitectura propuesta basada en Li, Ding y Sun genera predicciones más precisas obteniendo los mejores resultados de RMSE, RMSE early, RMSE mid y valores muy cercanos al mejor valor de RMSE late obtenido por el modelo **Ba_19_1024_250_1_3**.
3. Un análisis cualitativo de los gráficos de dispersión muestra que los modelos entrenados con los criterios mencionados anteriormente y con la arquitectura propuesta basada en Li, Ding y Sun ajustan con más precisión los valores de RUL estimado a los valores de RUL verdadero de los motores en el tramo early. Lo anterior se hace más evidente con un $N_{tw} = 19$.

Por otro lado, analizando los modelos entrenados con el dataset FD002+FD004 y con $N_{tw} = 19$ observamos que el modelo **Ba_19_512_250_2_4** obtiene el menor RMSE y RMSE late de todos los modelos. A su vez, el modelo **Ba_19_1024_500_2_4** obtiene un valor de RMSE bastante cercano al modelo anterior junto con el RMSE early y RMSE mid más bajos con los criterios mencionados anteriormente.

Repitiendo el mismo ejercicio anterior con $N_{tw} = 10$ se observa que el modelo **Ba_10_512_500_2_4** obtiene el menor valor de RMSE y RMSE mid. Por otro lado, el modelo **Ba_10_512_250_2_4** obtiene el menor valor de RMSE early y el modelo **Ba_10_1024_500_2_4** obtiene un valor de RMSE late comparable al mejor valor de esta métrica: el modelo **Li_10_512_250_2_4**. Usando el dataset FD002+FD004 es posible notar lo siguiente:

1. Un mayor tamaño de $N_{tw}$ no conlleva una mejora en la predicción de los modelos dado que sus valores RMSE aumentan también.
2. Ante los mismos hiperparámetros los modelos entrenados usando la arquitectura propuesta basada en Babu, Zhao y Li generan predicciones más precisas y obtienen mejores resultados de RMSE, RMSE early y RMSE mid en este dataset.
3. Un análisis cualitativo de los gráficos de dispersión muestra que los modelos entrenados con los criterios mencionados anteriormente y con la arquitectura propuesta basada en Babu, Zhao y Li ajustan con más precisión los valores de RUL estimado a los valores de RUL verdadero de los motores en el tramo early para ambos tamaños de $N_{tw}$.

Con el análisis anterior se deduce que la arquitectura propuesta basada en Li, Ding y Sun logra capturar y extraer características relevantes de mejor manera en las señales de los sensores del motor turbofan asociadas a una condición de operación (dataset FD001+FD003). La emulación de la convolución en una dimensión con sus filtros de ancho 1 permiten encontrar características relevantes independientes dentro de cada una de las 14 señales.

A su vez, la arquitectura propuesta basada en Babu, Zhao y Li logra capturar y extraer características relevantes de las señales de los sensores del motor turbofan asociadas a cada una de las seis condiciones de operación (dataset FD002+FD004). Se presume que el comportamiento de las señales en cada condición de operación es distinta en un mismo RUL, por lo que el primer filtro de la arquitectura que convoluciona con todas las señales es capaz de encontrar esas características y extraerlas de mejor manera que la arquitectura propuesta basada en Li, Ding y Sun.

Se observa también que ningún modelo obtiene valores de RMSE mid menores a 19,5 debido a que las arquitecturas propuestas no son capaces de extraer características relevantes en las señales cuando el RUL verdadero está en ese tramo. A diferencia del tramo mid, en el tramo late se obtienen mejores valores de RMSE gracias al preprocesamiento mencionado en la sección 6.1, en la cual se modifican los RUL > 120 a RUL = 120 facilitando a los modelos el poder mapear el valor de las señales de los sensores cuando el RUL es muy alto a un valor de RUL seteado por defecto.

Realizando un análisis cualitativo en los gráficos de training y validation error versus número de epochs de los modelos se puede observar lo siguiente:

- Ningún modelo muestra subajuste y/o sobreajuste. A mayor número de epochs ambas curvas de error describen una tendencia decreciente en todos los modelos.
- En todos los modelos la curva de validation error tiene un menor valor que la curva de training error. Dado que se usa la técnica de Dropout en ambas arquitecturas propuestas, la mitad de las neuronas en cada capa donde se aplica esta técnica queda deshabilitada en la etapa de entrenamiento. Lo anterior reduce el tamaño de la matriz de pesos de la capa, lo que flexibiliza la captura de representaciones complejas en los datos permitiéndole así al modelo poder generalizar para datos nuevos (en la etapa de validación) de mejor manera atenuando el sobreajuste. Al calcular el validation error todas las neuronas quedan habilitadas nuevamente, por lo que el modelo puede volver a capturar esas representaciones complejas, implicando obtener valores de validation error menores al training error en capa epoch. Si no es aplicado Dropout, un ejemplo de la gráfica de training y validation error versus número de epochs es de la siguiente forma (ver Figura 7.1):

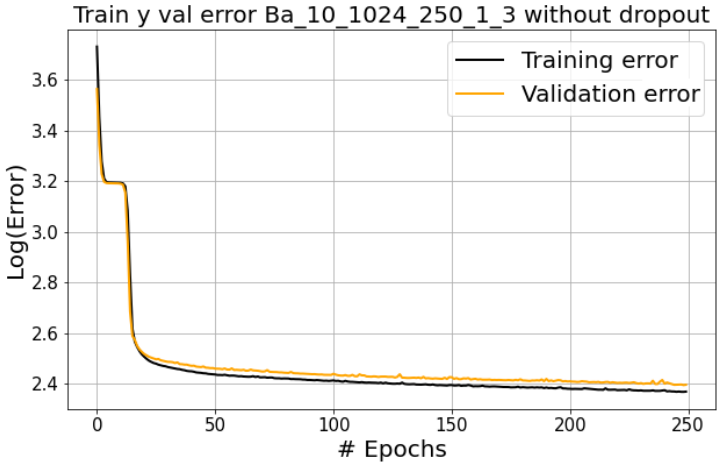



**Figura 7.1:** Gráfico training y validation error vs número de epochs modelo Ba_10_1024_259_1_3 sin Dropout
(Fuente: Elaboración propia)

Se observa en el gráfico anterior que el validation error es levemente superior al traning error del modelo **Ba_10_1024_259_1_3** sin Dropout, existiendo un leve sobreajuste.

Además, se destaca que gracias al preprocesamiento del Turbofan Engine Degradation Simulation Dataset, el uso de Adam como optimizador y la elección de hiperparámetros, la arquitectura propuesta basada en Babu, Zhao y Li obtiene valores de RMSE más bajos que los registrados en [16]: en el dataset FD001 y FD003 obtuvieron un RMSE de 18,44 y 19,81 respectivamente con un $N_{tw} = 15$ versus el RMSE de 17,75 del modelo **Ba_10_512_250_1_3** ($N_{tw} = 10$). Además en el dataset FD002 y FD004 obtuvieron un RMSE de 30,29 y 29,15 respectivamente con un $N_{tw} = 15$ versus el RMSE de 18,05 del modelo **Ba_10_512_500_2_4** ($N_{tw} = 10$).

Por último, también se destaca que la arquitectura propuesta basada en Li, Ding y Sun logra valores de RMSE iguales o mas bajos que los registrados en [15]: para el dataset FD001 con un $N_{tw} = 10$ obtuvieron un valor de RMSE de 16,9 versus el RMSE de 17 del modelo **Li_10_1024_250_1_3**. Para un $N_{tw} = 20$ obtienen un valor de RMSE de 16,5 versus el RMSE de 13,88 del modelo **Li_19_1024_250_1_3** ($N_{tw} = 19$). A su vez para el dataset FD002 y FD004 obtuvieron un valor de RMSE de 22,36 (para $N_{tw} = 20$) y 23,31 (para $N_{tw} = 15$) respectivamente versus el RMSE de 20,06 del modelo **Li_10_512_500_2_4**.

# 8 | Conclusiones y trabajo futuro

El mantenimiento predictivo permite a las organizaciones que lo adoptan un ahorro de recursos bastante importante. Por eso, esta área y sus técnicas para ejecutarlo son ampliamente estudiadas en la industria. Teniendo en cuenta su gran popularidad de los últimos años, es posible utilizar técnicas de aprendizaje automático para entrenar modelos que permitan planificar estos mantenimientos de mejor manera, dado que con ellos es posible predecir el *Remaining Useful Life* del componente o sistema que se desea monitorear de forma bastante precisa si se cuentan con los datos necesarios (entregados por sensores por ejemplo).

En ciertos casos no se cuenta con el RUL de forma directa como en el caso del Vega Shrink-wrapper Dataset, por lo que es necesario usar técnicas de aprendizaje automático para encontrar una función de degradación que permita cuantificar esta variable. Esta tarea se complica si el componente o sistema pasa por varios modos de operación de forma estocástica durante el tiempo que dure la serie de tiempo multivariada. Otro enfoque a lo anterior es entrenar modelos no supervisados de aprendizaje automático con los datos, pero es necesario confirmar la validez del modelo con un experto que lo certifique.

Un dataset popular de mantenimiento predictivo para el cual se han desarrollado diversas arquitecturas y modelos de aprendizaje profundo para predecir la falla de motores de avión es el Turbofan Engine Degradation Simulation Data Set.

Gracias al preprocesamiento de los datos de entrada a los modelos de CNN, elección de hiperparámetros, optimizadores, añadir técnicas como Dropout, capas convolucionales o de pooling se puede mejorar el desempeño de las métricas de evaluación de estos. Lo anterior es aplicado a dos arquitecturas propuestas en el estado de arte, cuantificando las mejorías de estos ajustes y eligiendo el mejor modelo de acuerdo a sus diferentes configuraciones.

En vista de los resultados y recordando que el tramo early es el más importante debido a su cercanía con la falla del motor de avión, se realizan las siguientes recomendaciones:

- Para un motor de avión con solo una condición de operación (dataset FD001+FD003) primero se recomienda utilizar el modelo **Li_10_1024_500_1_3** dado que es el modelo más preciso en el tramo de RUL early y solo se debe esperar una ventana de tiempo de $N_{tw} = 10$ para obtener una primera predicción. Luego de esperar una ventana de tiempo de $N_{tw} = 19$ del funcionamiento del motor, es posible obtener una predicción bastante más precisa del RUL en el tramo early utilizando el modelo **Li_19_512_500_1_3**. Con ambas predicciones se planifica y ejecuta el mantenimiento predictivo.
- Para un motor de avión con las seis condiciones de operación (dataset FD002+FD004) solo se recomienda utilizar el modelo **Ba_10_512_250_2_4** ya que es el modelo más preciso en el tramo de RUL early, incluso con una ventana de tiempo de $N_{tw} = 10$. Luego esta predicción se toma en cuenta para planificar y ejecutar el mantenimiento predictivo correspondiente.

Por último, con las arquitecturas y las configuraciones propuestas es posible mejorar las métricas de desempeño de RMSE de los modelos propuestos por Babu, Zhao y Li [16] con un $N_{tw}$ menor al usado por ellos. Además, es posible igualar la métrica de desempeño de RMSE del modelo propuesto por Li, Ding y Sun [15] usando el dataset FD001 con un $N_{tw} = 10$ y mejorarla para un $N_{tw} = 19$. A su vez, usando los datasets FD002 y FD004 es posible mejorar la métrica de desempeño de RMSE de los modelos propuestos en [15] con un $N_{tw}$ menor al usado por ellos.

## 8.1. Trabajo futuro

En esta memoria se utilizan arquitecturas basadas en CNN (mencionadas en el estado del arte) para cumplir con el objetivo general y objetivos específicos vistos en la sección 1.5, pero también existen otros tipos de arquitecturas basadas en redes neuronales recurrentes o RNN que se utilizan ampliamente en series de tiempo multivariadas. Según [22] una RNN procesa secuencias iterando a través de los elementos de esta, manteniendo un **estado** que contiene información relativa de lo que ha visto hasta ahora. El uso de ese estado le permite a estas arquitecturas encontrar representaciones complejas dentro de las señales en una serie de tiempo multivariada como es el caso de esta memoria. Se ha omitido el uso de RNN dado que la arquitectura propuesta por Li, Ding y Sun en [15] es comparada con una arquitectura basada en RNN y otra en LSTM, obteniendo una mejor métrica de RMSE que estas.

Tomando en cuenta lo anterior y pensando en las posibilidades de mejora de esta memoria, se proponen las siguientes investigaciones y desarrollos:

- Investigar y/o desarrollar una arquitectura basada en RNN que repita la metodología expuesta en esta memoria y que pueda igualar o mejorar las métricas expuestas.
- Repetir la metodología expuesta en esta memoria con un dataset real de motores de avión o con otro tipo de componente y/o sistema que requiera mantenimiento predictivo, en el cual se tenga disponible un dataset junto con un valor de RUL asociado a cada muestra del mismo para aplicar un algoritmo de aprendizaje profundo supervisado.
- Ocupar otra configuración de hiperparámetros, tamaños de $N_{tw}$ y adición de capas (convolucionales, pooling y/o fully-connected) para encontrar modelos que obtengan un mejor desempeño en las métricas de RMSE a los descritos en este trabajo de memoria de titulación.
- Desarrollar un sistema que permita pasar a producción los modelos recomendados, para así obtener en tiempo real una medida del RUL que pueda ser utilizado por las partes interesadas de una organización que trabaje con estos motores.

# Bibliografía

[1] R.K. Mobley, *An Introduction to Predictive Maintenance*. 2nd edition. Butterworth-Heinemann, 2002, pp. 1-2. 1.1

[2] Persistence Market Research. (2018). Global Market Study on Safety Sensors and Switches: Significant Demand for Safety Sensors and Switches for Leak Detection to be Observed in the Coming Years [Online]. https://www.persistencemarketresearch.com/market-research/safety-sensors-and-switches-market.asp [consulta: 03-03-2020]. 1.1

[3] R. Gouriveau, K. Medjaher, N. Zerhouni, *From Prognostics and Health Systems Management to Predictive Maintenance 1: Monitoring and Prognostics*. 1st edition. John Wiley & Sons, 2016, p. 4. 1.2

[4] D. Banjevic, "Remaining useful life in theory and practice". *Metrika*, vol. 69, pp. 337-349. December 2008. 1.2.1

[5] BigData Republic. (2017). Machine learning for predictive maintance. where to start?. [Online]. https://medium.com/bigdatarepublic/machine-learning-for-predictive-maintenance-where-to-start-5f3b7586acfb [consulta: 09-08-2020]. 1.2

[6] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*. MIT Press, 2016, chapter 5, http://www.deeplearningbook.org 1.3.1, 5.5

[7] J. Wang, *Data Mining: Opportunities and Challenges*. Idea Group Inc, 2003, p. 261. 1.3.1

[8] Tristan Greene. (2020). 2010 – 2019: The rise of deep learning. [Online]. https://thenextweb.com/artificial-intelligence/2020/01/02/2010-2019-the-rise-of-deep-learning/ [consulta: 23-03-2020]. 1.3.1

[9] F. Chollet, "Evaluating machine-learning models", en *Deep Learning with Python*. 1st edition. Manning, 2017, pp. 97-99. 1.3.2

[10] T. Hanzák, "Methods for periodic and irregular time series", Ph.D. thesis, Charles University, Prague, 2014. 1.4.1

[11] D. J. Wright, "Forecasting Data Published at Irregular Time Intervals Using an Extension of Holt's Method". *Management Science*, vol. 32, no. 4, pp. 499-510. April 1986. 1.4.1

[12] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*. MIT Press, 2016, chapter 8, http://www.deeplearningbook.org 2.6, 2.6

[13] D. P. Kingma, J. L. Ba, "Adam: A Method for Stochastic Optimization". *arXiv:1412.6980 [cs]*, December 2014. 2.6

[14] Z. Tian, "An artificial neural network approach for remaining useful life prediction of equipments subject to condition monitoring". *2009 8th International Conference on Reliability, Maintainability and Safety*, pp. 143-148. July 2009. 3

[15] X. Li, Q. Ding, J. Sun, “Remaining useful life estimation in prognostics using deep convolution neural networks". *Reliability Engineering & System Safety*, vol. 172, pp. 1-11. April 2018. 3, 3, 5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.2, 5.3, 5.4, 5.5, 5.6, 7, 8, 8.1

[16] G. Babu, P. Zhao, X.-L. Li, “Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life". *Lecture Notes in Computer Science*, vol. 9642, pp. 214-228. March 2016. 3, 5.1.1, 5.1.3, 5.1.4, 5.4, 5.5, 5.5, 5.6, 7, 8

[17] F. Chollet, ”Sequence processing with convnets", en *Deep Learning with Python*. 1st edition. Manning, 2017, pp. 225-232. 3

[18] W. Caesarendra, T. Tjahjowidodo, ”A Review of Feature Extraction Methods in Vibration-Based Condition Monitoring and Its Application for Degradation Trend Estimation of Low-Speed Slew Bearing”, *Machines*, vol. 5, no. 4, p. 21, September 2017. 4.1.1

[19] J. Sola, J. Sevilla, ”Importance of input data normalization for the application of neural networks to complex industrial problems”, *Nuclear Science, IEEE Transactions on*, vol. 44, no. 3, pp. 1464 - 1468, June 1997. 5.1.2

[20] P. Radiuk, ”Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets”, *Information Technology and Management Science*, vol. 20, no. 1, pp. 20-24, December 2017. 5.2

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, ”Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929-1958, June 2014. 5.3

[22] F. Chollet, ”Understanding Recurrent Neural Networks", en *Deep Learning with Python*. 1st edition. Manning, 2017, p. 196. 8.1

# A | Códigos

Los Colab Notebooks utilizados en el desarrollo de este trabajo de memoria de titulación se encuentran en el siguiente repositorio de Github:

- https://github.com/farayal/memoria_turbofan [última consulta: 13-10-2020]