

**UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA**  
**SEDE VIÑA DEL MAR – JOSÉ MIGUEL CARRERA**

**PROPUESTA DE MEJORA A GESTIÓN DE MANTENIMIENTO UTILIZANDO  
HERRAMIENTAS DE APRENDIZAJE SUPERVISADO EN EQUIPO CRÍTICO DE  
PLANTA DE CELULOSA**

Trabajo de Titulación para optar al  
Título de Ingeniero en MANTENIMIENTO  
INDUSTRIAL LICENCIADO EN  
INGENIERÍA

Alumno:

Luis Felipe Gálvez Bernal

Profesor Guía:

Mg. Ing. Andrés Aránguiz Garrido

## **DEDICATORIAS**

*Dedicado a Jacqueline, a José Luis, y a todos los que ya no están con nosotros de nuestra familia. Gracias totales por todo el apoyo recibido desde siempre.*

*Quisiera agradecer a mis compañeros de carrera por los buenos momentos y también a todos los profesores involucrados en mi formación profesional como uno de los ingenieros del mañana.*

## **RESUMEN**

**KEYWORDS:** Mantenimiento Predictivo, Machine Learning, Aprendizaje Supervisado, Redes Neuronales Artificiales, Pronóstico, Eventos de Fallas.

Cuando se utiliza machine learning (ML) para mantenimiento predictivo (PdM), es posible lograr pronósticos de eventos de fallas lo que permite mejorar la gestión de mantenimiento, resulta un tema interesante porque es una rama de la Inteligencia Artificial (IA). El presente trabajo de titulación se enfocará en un equipo crítico de una planta de celulosa en Chile.

Se realiza una investigación del estado del arte sobre el uso de herramientas de ML en el mantenimiento industrial. Analizando algunos artículos de revistas científica como bibliografía disponible. Luego se diferencian los tipos de herramientas y que tipo de problemas resuelven en la industria. La metodología consiste en aplicar un modelo que combina un modelo Auto Regresivo de Medias Móviles (ARMA) con Redes Neuronales Artificiales (RNA) aplicado en serie. Existe evidencia que al combinar estas herramientas se puede apalancar los resultados obtenidos de ARMA, logrando buenos resultados.

Al analizar un histórico de fallas de la planta de celulosa que considera 3 años de registros, se determina que en una primera instancia que el modelo de vida Weibull tiene una métrica de precisión RMSE de validación de 673,2 tras obtener sus parámetros y generar eventos aleatorios. También se determina que el modelo ARMA que minimiza el valor de RMSE es un modelo ARMA(1,0) o AR(1) con un RMSE de validación de 137,1. La estructura de una RNA puede ser arbitraria, por lo que se decide proponer un modelo base de 2 entradas, 5 capas, 10 neuronas por capa. Se obtiene un RMSE de validación de 136,2 que durante el proceso de entrenamiento del modelo converge muy rápido.

Se realizan 624 iteraciones para encontrar el mejor modelo que minimice el valor de RMSE para su validación. Considerando cambios sobre el número de entradas, introduciendo nuevas entradas con valores de Confiabilidad o  $R(t)$ , Densidad de Probabilidad de Fallas o  $f(t)$  junto con la Tasa de Fallas o  $\lambda(t)$ . El modelo que optimiza los resultados anteriores es el modelo AR(1)+RNA(2,64,64,1) de 4.417 parámetros, que tiene de entrada valores de  $R(t)$  y  $\lambda(t)$ , los que se obtienen tras modelar los datos desde el histórico de fallas y evaluar con los tiempos de los pronósticos del modelo ARMA. Resulta el modelo n° 10.5 con un RMSE de validación de 33,9. El cual es excelente para pronosticar eventos de corto plazo, casi exacto para eventos de mediano plazo y predice con 200 horas de anticipación los eventos de largo plazo. Supera en un 94,9% al modelo de vida Weibull, supera al modelo AR(1) en 75,3% y supera el modelo base en un 75,1%.

ÍNDICE

INTRODUCCIÓN ..... 1

OBJETIVOS..... 2

CAPITULO 1: MACHINE LEARNING EN MANTENIMIENTO PREDICTIVO..... 3

1. MACHINE LEARNING EN MANTENIMIENTO PREDICTIVO..... 4

1.1. MANTENIMIENTO EN PLANTA DE CELULOSA ..... 4

1.1.1. Modelo de vida Weibull .....4

1.2. PROBLEMÁTICA..... 5

1.3. INTELIGENCIA ARTIFICIAL Y MACHINE LEARNING ..... 6

1.4. ESTADO DEL ARTE ..... 7

1.4.1. Redes Neuronales (RN).....8

1.4.2. Random Forests (RF) .....8

1.4.3. Support Vector Machine (SVM) .....8

1.4.4. K-means.....8

1.5. ENFOQUE DEL TRABAJO ..... 8

1.6. MODELO ARIMA ..... 9

1.6.1. Modelo ARIMA en Python .....10

1.7. REDES NEURONALES ARTIFICIALES ..... 11

1.7.1. Neurona Biológica.....12

1.7.2. Tipos de Redes Neuronales .....13

1.7.3. Aplicaciones de las RNA.....13

1.7.4. Ventajas y Desventajas de las RNA .....14

1.8. FUNCIONES DE ACTIVACIÓN..... 15

1.8.1. ReLU (Rectified Logic Unit).....15

1.8.2. Leaky ReLU: .....15

1.8.3. Sigmoides.....16

1.8.4. Tangente Hiperbólica .....17

1.9. ENTRENAMIENTO DE REDES NEURONALES ..... 17

1.9.1. Iniciación de los parámetros .....18

1.9.2. Retro propagación del Error (Back propagation) .....18

1.9.3. Descenso de Gradiente Estocástico .....18

1.9.4. Momentum .....19

1.9.5. Regla de la Cadena .....20

1.10. ESTRUCTURA DE REDES NEURONALES ..... 21

1.10.1. Perceptrón.....22

1.10.2. Neurona Monocapa .....22

1.10.3. Red Neuronal Monocapa .....24

1.10.4. MLP para Regresión.....25

1.10.5. MLP para Clasificación.....25

CAPÍTULO 2: FORMULACIÓN DE UN MODELO DE ML ..... 27

2. FORMULACIÓN DE UN MODELO DE ML ..... 28

2.1. CONJUNTO DE DATOS..... 28

2.2. MODELO ML CONTRA DISTRIBUCIÓN WEIBULL..... 29

2.2.1. Confiabilidad con Distribución Weibull .....29

|                                                         |                                         |           |
|---------------------------------------------------------|-----------------------------------------|-----------|
| 2.3.                                                    | MODELO PREDICTIVO DE ML: ARIMA + RNA:   | 32        |
| 2.3.1.                                                  | Código de Modelo ARIMA                  | 33        |
| 2.3.2.                                                  | Código de Modelo de RNA                 | 35        |
| 2.3.3.                                                  | Resultados Obtenidos del Modelo Base    | 36        |
| <b>CAPÍTULO 3: EVALUACIÓN DE UN MODELO ÓPTIMO DE ML</b> |                                         | <b>39</b> |
| 3.                                                      | EVALUACIÓN DE UN MODELO ÓPTIMO DE ML    | 40        |
| 3.1.                                                    | CAMBIOS EN EL MODELO BASE               | 40        |
| 3.1.1.                                                  | Iteración sobre Modelo Base             | 40        |
| 3.1.2.                                                  | Procedimiento de Iteración              | 40        |
| 3.1.3.                                                  | Formato de Planilla de Iteración        | 41        |
| 3.1.4.                                                  | Iteraciones sobre Modelos               | 44        |
| 3.1.5.                                                  | Resultados de la Iteración              | 44        |
| 3.2.                                                    | MODELO FINAL PROPUESTO                  | 47        |
| 3.3.                                                    | ANÁLISIS DE COSTOS E IMPACTO DEL MODELO | 48        |
| <b>CONCLUSIONES</b>                                     |                                         | <b>49</b> |
| <b>BIBLIOGRAFÍA</b>                                     |                                         | <b>50</b> |
| <b>ANEXO A</b>                                          |                                         | <b>51</b> |
| <b>ANEXO B</b>                                          |                                         | <b>52</b> |
| <b>ANEXO C</b>                                          |                                         | <b>53</b> |
| <b>ANEXO D</b>                                          |                                         | <b>54</b> |
| <b>ANEXO E</b>                                          |                                         | <b>57</b> |
| <b>ANEXO F</b>                                          |                                         | <b>59</b> |
| <b>ANEXO G</b>                                          |                                         | <b>62</b> |

**ÍNDICE DE FIGURAS**

|              |                                                           |    |
|--------------|-----------------------------------------------------------|----|
| Figura 1-1:  | Curva de la Bañera típica de la Tasa de Fallas            | 5  |
| Figura 1-2:  | Forma de programar ML                                     | 6  |
| Figura 1-3:  | Nodo o Neurona                                            | 11 |
| Figura 1-4:  | Red neuronal (6, 4, 5, 4, 1)                              | 12 |
| Figura 1-5:  | Partes de una Neurona Biológica                           | 13 |
| Figura 1-6:  | Función de Activación ReLU                                | 15 |
| Figura 1-7:  | Función de Activación Leaky ReLU                          | 16 |
| Figura 1-8:  | Función de Activación Sigmoide                            | 16 |
| Figura 1-9:  | Función de Activación Tangente Hiperbólica                | 17 |
| Figura 1-10: | Proceso de Momentum                                       | 20 |
| Figura 1-11: | Estructura RNA Feed-forward en problemas de clasificación | 21 |

|                                                                         |    |
|-------------------------------------------------------------------------|----|
| Figura 1-12: Ejemplo de Neurona Monocapa. ....                          | 23 |
| Figura 1-13: Ejemplo de Red Neuronal Monocapa. ....                     | 24 |
| Figura 1-14: Perceptrón Multicapa para Problemas de Regresión.....      | 25 |
| Figura 1-15: Perceptrón Multicapa para Problemas de Clasificación.....  | 26 |
| Figura 2-1: Gráfico Tiempos Hasta la Falla.....                         | 28 |
| Figura 2-2: Diagrama de Flujo del código “EjecutaWEIBULL”.....          | 30 |
| Figura 2-3: Gráfico de Dispersión Weibull.....                          | 30 |
| Figura 2-4: Tasa de Fallas y Confiabilidad de los Datos Reales.....     | 31 |
| Figura 2-5: Histograma de los Datos Simulados en Python. ....           | 32 |
| Figura 2-6 : Diagrama de Flujo del código “EjecutaBROWNLEEbased”. ....  | 33 |
| Figura 2-7: Diagrama de Flujo del código “EjecutaARMA1”.....            | 34 |
| Figura 2-8: Conjunto de Validación del Modelo ARIMA (1,0,0). ....       | 34 |
| Figura 2-9 : Diagrama de Flujo del código “EjecutaRNA”. ....            | 35 |
| Figura 2-10: Validación del Modelo AR(1)1+RNA(5,10,10,10,10,10,1). .... | 37 |
| Figura 2-11: Data Real vs. Predicción en Modelo Base. ....              | 38 |
| Figura 3-1: Validación Modelo 10.5 AR1+RNA(2,64,64,1). ....             | 45 |
| Figura 3-2: Data Real vs. Predicción en Modelo 10.5. ....               | 45 |
| Figura 3-3: Validación Modelo 12.6 AR1+RNA(2,64,64,1). ....             | 46 |
| Figura 3-4: Data Real vs. Predicción del Modelo . ....                  | 47 |
| Figura 3-5: Ejemplo Red Neuronal Propuesta. ....                        | 48 |

**ÍNDICE DE TABLAS**

|                                                                  |    |
|------------------------------------------------------------------|----|
| Tabla 1-1: Aplicaciones de las RNA.....                          | 14 |
| Tabla 1-2:Ventajas y Desventajas de las RNA. ....                | 14 |
| Tabla 2-1: Resultados Estimación Parámetros Weibull.....         | 31 |
| Tabla 2-2: Resultados RMSE.....                                  | 38 |
| Tabla 3-1: Primera iteración sobre las entradas del modelo. .... | 41 |
| Tabla 3-5: Formato de Planilla de Iteración sobre Modelos. ....  | 43 |

## **INTRODUCCIÓN**

Actualmente existen herramientas de machine learning (ML) que están basadas en la biología neuronal: las Redes Neuronales Artificiales (RNA). En la cual se hace una analogía de su funcionamiento para crear sistemas inteligentes que puedan realizar predicciones de eventos de fallas.

En la industria forestal se está trabajando para poder anticiparse a posibles fallas que puedan impactar en su proceso productivo. Lo que trae costos asociados por el mantenimiento correctivo o preventivo realizado. Lograr la predicción de eventos de fallas permitiría generar un ahorro considerable respecto a costos de ineficiencia.

Con ML se puede elaborar modelos de buena precisión que permitirían mejorar la gestión de ciertos modos de fallas, facilitando la gestión temprana del mantenimiento tras un pronóstico basado en históricos de fallas. Integrar herramientas de aprendizaje supervisado de ML permite la innovación dentro de los procesos de mantenimiento, ya que forma parte de la Inteligencia Artificial aplicada en la industria.

Inicialmente se puede considerar la formulación arbitraria de un modelo de redes neuronales artificiales, la cual no siempre puede ser la mejor. Generar iteraciones sobre la arquitectura de una RNA permite optimizar los parámetros para encontrar el mejor modelo acorde a los datos analizados para su entrenamiento y posterior validación.

Python permite importar librerías que optimizan el código elaborado, agilizando el proceso de análisis de datos en el mantenimiento industrial. Es una herramienta excelente cuando se dispone de datos masivos en algún histórico de fallas.

La lógica de ML consiste en entrenar modelos con los datos disponibles y sus resultados, el modelo debe aprender de los datos y ser capaz de definir que reglas seguir para obtener los resultados con los que se le entrenó. Luego de eso se debe validar el modelo mediante la evaluación de desempeño obtenido y su testeo en una muestra de los datos para demostrar su aplicación efectiva.

## **OBJETIVOS**

### **OBJETIVO GENERAL**

Proponer mejoras a la gestión de mantenimiento de Celulosa Arauco, mediante la aplicación de un modelo de aprendizaje supervisado de redes neuronales artificiales, para el pronóstico de eventos de fallas en un equipo crítico.

### **OBJETIVOS ESPECÍFICOS**

- Comparar herramientas de machine learning utilizadas en mantenimiento industrial, para la selección de un modelo/algoritmo, mediante la diferenciación de éstos.
- Formular un modelo de aprendizaje supervisado, mediante la integración de algunos cambios, para la evaluación de su desempeño.
- Evaluar modelos con diferentes arquitecturas, mediante pruebas iterativas, para la propuesta de un modelo adecuado a los datos analizados.

## **CAPÍTULO 1: MACHINE LEARNING EN MANTENIMIENTO PREDICTIVO**

## 1. MACHINE LEARNING EN MANTENIMIENTO PREDICTIVO

El avance de la tecnología ha hecho que las industrias evolucionen, logrando aplicar inteligencia artificial (IA) dentro de sus procesos productivos. El mantenimiento industrial no se queda atrás, logrando conocimientos como predicción de eventos de fallas o identificación temprana de fallas para mejorar la gestión del mantenimiento.

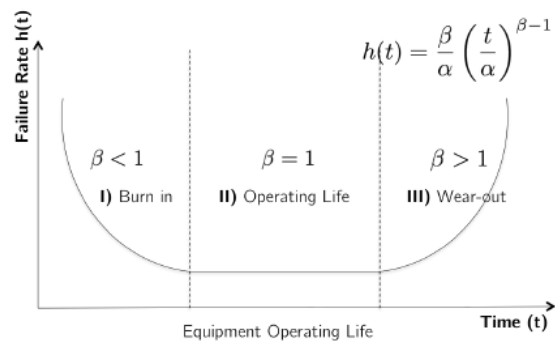
### 1.1. MANTENIMIENTO EN PLANTA DE CELULOSA

En Celulosa Arauco el mantenimiento industrial es una de las áreas importantes para apoyar los procesos productivos de la organización. Según información facilitada, el costo por falla en la línea n°1 (costo de ineficiencia) es de 7.694,4 [USD/Hr], lo que implica una pérdida directa por el tiempo fuera de servicio que se encuentre algún equipo de la línea de producción, afectando a los resultados financieros.

#### 1.1.1. Modelo de vida Weibull

Los eventos de fallas ocurren de forma aleatoria y pueden ser modelados mediante la distribución Weibull debido a su versatilidad ya que puede cambiar su forma. El procedimiento consiste en tomar una muestra grande de los datos de algún histórico de fallas para modelar, obtener sus parámetros y lograr una generalización de la población (tiempo hasta la falla). Los parámetros del modelo son  $\alpha$  de vida característica y  $\beta$  de forma (Mora, 2009).

En el área de mantenimiento se utilizan distribuciones de probabilidad para caracterizar el comportamiento de fallas en equipos. Normalmente se ocupa la distribución Exponencial para tasa de fallas constante, y la Weibull o Log-Normal para tasa de fallas variables. En la Figura 1-1 se muestra la curva de la bañera típica de la tasa de fallas, cuando beta es menor a 1 el equipo se encuentra en zona de rodaje o mortalidad infantil, si beta es igual a 1 el equipo se encuentra en zona de fallas aleatorias con tasa de fallas constante, y cuando beta es mayor a 1 el equipo se encuentra en zona de desgaste. Se muestra también la ecuación de la tasa de fallas. Por otro lado, en las ecuaciones (1), (2), (3) y (4) corresponden a la función de densidad de probabilidad de fallas, función de distribución acumulada de fallas, función de confiabilidad y función de esperanza o tiempo medio entre fallas (MTBF).



Fuente: Forecasting Fault Events for Predictive Maintenance using Data-driven Techniques and ARMA Modeling (Baptista, Sankararaman, de Medeiros, Nascimento Jr., & Henriquesa, 2018).

Figura 1-1: Curva de la Bañera típica de la Tasa de Fallas.

$$f(t) = \frac{\beta}{\alpha} \left(\frac{t}{\alpha}\right)^{\beta-1} e^{-\left[\frac{t}{\alpha}\right]^\beta} \tag{1}$$

$$F(t) = 1 - e^{-\left[\frac{t}{\alpha}\right]^\beta} \tag{2}$$

$$R(t) = e^{-\left[\frac{t}{\alpha}\right]^\beta} \tag{3}$$

$$E[t] = MTBF = \alpha \Gamma\left(1 + \frac{1}{\beta}\right) \tag{4}$$

1.2. PROBLEMÁTICA

La desventaja del modelo Weibull es que no es preciso ya que solamente está basado en el modelado estadístico de datos. Por ejemplo, la esperanza o MTBF en una distribución Weibull es la generalización del promedio de los datos de la población modelada. El promedio poblacional no permite una anticipación efectiva a un evento de falla.

La detección temprana de los eventos de fallas podría facilitar la programación efectiva de tareas de mantenimiento que a la vez permitan generar ahorros debido al costo de ineficiencia que corresponden a 7.694,4 [USD/Hr].

1.3. INTELIGENCIA ARTIFICIAL Y MACHINE LEARNING

La Inteligencia Artificial (IA) nació en los 50s, cuando un grupo de pioneros en el área emergente de ciencias computacionales se preguntaban cómo los computadores podrían “pensar”. Una definición breve del área sería: el esfuerzo de automatizar las tareas intelectuales que normalmente realizan los humanos (Chollet, 2018).

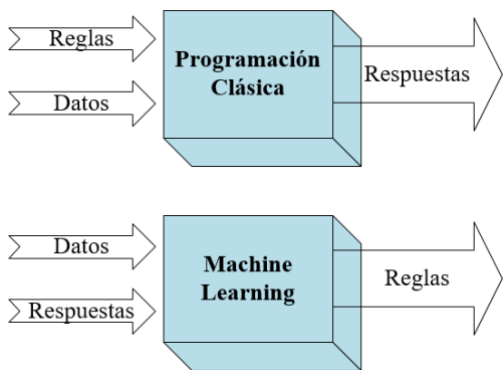
Existen varios elementos que componen la IA, dentro de los cuales se pueden encontrar tres grandes ramas. Una de ellas corresponde a las Redes Neuronales Artificiales (RNA), que a su vez forman parte de técnicas de aprendizaje supervisado en el aprendizaje automático o Machine Learning (ML).

ML surge desde las preguntas:

- Podría un computador ir más allá de lo que sabemos de ordenarle que haga?
- Podría aprender por sí mismo el cómo realizar una tarea específica?
- En vez de programar las reglas, podría un computador aprender dichas reglas observando los datos?

ML proporciona enfoques poderosos para aplicaciones de mantenimiento predictivo (PdM). Sin embargo, el desempeño de esas aplicaciones depende de la elección apropiada de la técnica de ML (Carvalho, y otros, 2019).

En la programación clásica, los humanos introducen las reglas y los datos para ser procesados y obtener respuestas. Mientras que en ML, se introducen datos como respuestas esperadas desde la data, para obtener las reglas o programa (Chollet, 2018). En la Figura 1-2 se ilustra lo anterior.



Fuente: Traducción, Deep Learning with Python (Chollet, 2018).

Figura 1-2: Forma de programar ML.

Para realizar ML se necesitan tres cosas:

- **Datos de entradas:** Si la tarea es predecir eventos de falla, podrían ser históricos de fallas de un equipo del proceso productivo en una empresa.
- **Ejemplos de salidas esperadas:** Podrían ser los registros de tareas correctivas en un histórico de fallas.
- **Una forma para medir si el algoritmo funciona bien:** Se utiliza como señal de retroalimentación para ajustar la forma en que el algoritmo funciona.

Existen dos tipos de problemas (o tareas) resueltos con un algoritmo de ML de Aprendizaje Supervisado, en relación al mantenimiento industrial se encuentran a continuación. Además, hay herramientas de ML que resultan versátiles, resolviendo ambos tipos de problemas.

- **Clasificación:** Tiene como objetivo predecir variables aleatorias discretas. Ejemplo: Identificación de estados de fallas en algún equipo o proceso.
- **Regresión:** Tiene como objetivo predecir variables aleatorias continuas. Ejemplo: Pronóstico de eventos de fallas en algún equipo o proceso.

#### 1.4. ESTADO DEL ARTE

En (Baptista, Sankararaman, de Medeiros, Nascimento Jr., & Henriquesa, 2018) compara un número de enfoques de inteligencia artificial (Redes Neuronales y otras cuatro más) contra el enfoque estadístico del modelado de datos de fallas con distribución Weibull, conocido como modelo de uso de vida restante (Remaining life usage model). La novedad de su trabajo es la integración de la metodología Autorregresivo de Medias Móviles (ARMA) con algoritmos de ML para predecir el próximo evento de falla basado en experiencias basadas.

Lamentablemente sus resultados indican que la única técnica de ML con el peor desempeño fueron las redes neuronales. Debido a las características del conjunto de datos utilizado que tiende a un sobreajuste (overfit), resultando en predicciones erróneas.

En (Carvalho, y otros, 2019) se realiza una revisión sistemática de ML aplicado a PdM. Tiene como objetivo cubrir las principales soluciones publicadas de técnicas de mantenimiento predictivo basado en métodos de machine learning. Dicho paper compara cuatro algoritmos más populares, los cuales se mencionan a continuación.

#### 1.4.1. Redes Neuronales (RN)

Es uno de los más comunes y aplicado de los algoritmos de ML. Se basan en el funcionamiento de neuronas biológicas. Ha sido utilizado para el reconocimiento de estados de falla o saludables en diversos equipos, por ejemplo, turbinas eólicas y equipos de producción de ánodos.

#### 1.4.2. Random Forests (RF)

El método es complejo, y toma más tiempo computacional comparado con otras técnicas de machine learning. Crea un bosque con múltiples árboles de decisión aleatorios. Tiene buen desempeño cuando el número de variables es mayor al número muestral.

#### 1.4.3. Support Vector Machine (SVM)

Es otro método de ML utilizado ampliamente y conocido debido a su alta precisión. Presenta dificultad al entender e interpretar el modelo. Ha sido aplicado en reconocimiento de estados de falla en diversos equipos industriales.

#### 1.4.4. K-means

Es un modelo de los algoritmos de clustering más utilizados que utiliza una estrategia de aprendizaje no supervisado para determinar k particiones o clusters desde un conjunto de datos. Aunque es fácil de implementar y presenta un buen desempeño y puede manejar gran conjunto de datos. Hay dificultad de encontrar el número de clusters y el algoritmo es sensible a la escala de datos de entrada.

### 1.5. **ENFOQUE DEL TRABAJO**

El presente trabajo de titulación se basa en el modelo descrito anteriormente, pero con la siguiente propuesta: primero entrenar un modelo Autorregresivo Integrado de Medias Moviles (ARIMA) con parámetro de Integración igual a 0, y luego en serie entrenar diferentes arquitecturas de Redes Neuronales Artificiales (RNA).

Se escoge RNA debido a su gran popularidad y su grado de complejidad. Además, debido a que el tema principal trata sobre ML, se explica con detalle más sobre ellas.

## 1.6. MODELO ARIMA

La teoría de modelos ARIMA ha sido desarrollada por muchos investigadores y su amplia aplicación fue debido al trabajo de Box y Jenkins en 1994 quienes elaboraron un modelo práctico y sistemático.

El modelo general ARIMA (p,d,q) es un modelo auto regresivo integrado de medias móviles de orden p, d, q. Un modelo ARMA(p,q) es equivalente a un modelo ARIMA(p,0,q).

La parte auto regresiva se denota como en la ecuación (5).

$$X_t = c + \sum_{i=1}^p \phi_i X_{t-i} + \varepsilon_t \quad (5)$$

Donde:

$\phi_i, \dots, \phi_p$ : parámetros del modelo.

$c$ : constante.

$\varepsilon_t$ : término error.

Por otro lado, la parte de medias móviles se denota como en la ecuación (6).

$$X_t = \varepsilon_t + \sum_{i=1}^q \theta_i \varepsilon_{t-i} \quad (6)$$

Donde:

$\theta_i, \dots, \theta_q$ : parámetros del modelo.

$\varepsilon_t, \varepsilon_{t-1}, \dots, \varepsilon_{t-q}$ : término error.

Además, tal como se indica en (Ramos, Oliveira, & Silva, 2014), la forma de una red neuronal de tres capas (considerando 2 ocultas y 1 de salida) que se representa como en la ecuación (7), es equivalente a un modelo no lineal auto regresivo (AR) o modelo ARIMA(p,0,0).

$$y_t = \alpha_0 + \sum \alpha_j f \left( \sum \beta_{ij} y_{t-i} + \beta_{0j} \right) + \varepsilon_t \quad (7)$$

Donde:

$y_i$ : Entradas (primera capa).

$\beta_{ij}$ : Pesos (primera capa).

$\beta_0$ : Sesgo (primera capa).

$a_j$ : Pesos (segunda capa).

$\alpha_0$ : Sesgo (segunda capa).

$f(\cdot)$ : Función de Activación.

$\varepsilon_t$ : Error.

En la siguiente sección se profundiza más sobre las redes neuronales artificiales, su aplicación, tipos, forma de entrenar y arquitectura.

#### 1.6.1. Modelo ARIMA en Python

Se propone utilizar como base lo propuesto en (Brownlee, 2021) que mediante un modelo ARIMA realiza una iteración triple para encontrar los parámetros del modelo para resolver problemas de pronóstico de series temporales en Python.

Consideraciones del código:

- La iteración se realiza sobre las variables p, d, q definidas previamente.
- El tamaño de entrenamiento es del 66% del total de una base de datos.
- Realiza 1 predicción a la salida del modelo.

En concordancia con el modelo de (Baptista, Sankararaman, de Medeiros, Nascimento Jr., & Henriquesa, 2018) y según las variables definidas en el código, el modelo devuelve como resultado ciertas métricas de precisión que logren minimizar el error como lo son: Raíz del error cuadrático medio o MSE, Error absoluto mediana o MAE, Error ponderado absoluto medio o MAPE y la desviación estándar muestral o SSD. La métrica de precisión seleccionada es RMSE lo que permite evaluar cada modelo.

Las ecuaciones correspondientes a las métricas de precisión mencionadas se encuentran en el Anexo A.

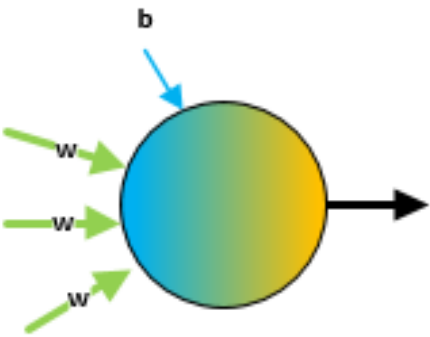
1.7. REDES NEURONALES ARTIFICIALES

Se les llama también como perceptrón multicapa (MLP) debido a que tienen múltiples capas con un cierto número de neuronas cada una, logrando superar las limitaciones del perceptrón. La historia asociada a las RNA está señalada en el Anexo B.

Las redes neuronales se pueden definir como aproximadores no lineales cuya estructura se basa en principios observados en los sistemas nerviosos de humanos y animales (Ponce Cruz, 2010). Otros autores también indican que las redes neuronales hacen referencia a la neurobiología, aunque algunos de los conceptos de aprendizaje profundo se desarrollaron inspirándose en nuestra comprensión del cerebro (Chollet, 2018).

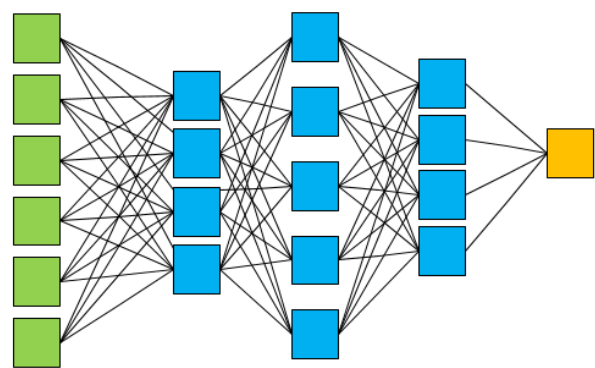
Además, una red neuronal se compone de varias unidades de procesamiento (nodos o neuronas) que tienen una operación relativamente simple. Esas unidades son generalmente conectadas por canales de comunicación que tienen un asociado peso (y por cada neurona un sesgo); y ellos solo operan sus datos locales que están indicados a través de sus conexiones (Carvalho, y otros, 2019). En la

Figura 1-3 se ilustra las conexiones que tiene un nodo o neurona y en la Figura 1-4 se muestra una red neuronal tipo feed-forward.



Fuente: Elaboración Propia, basado en Inteligencia Artificial con Aplicaciones a la Ingeniería (Ponce Cruz, 2010).

Figura 1-3: Nodo o Neurona.



Fuente: Elaboración Propia, basado en Inteligencia Artificial con Aplicaciones a la Ingeniería (Ponce Cruz, 2010).

Figura 1-4: Red neuronal (6, 4, 5, 4, 1).

1.7.1. Neurona Biológica

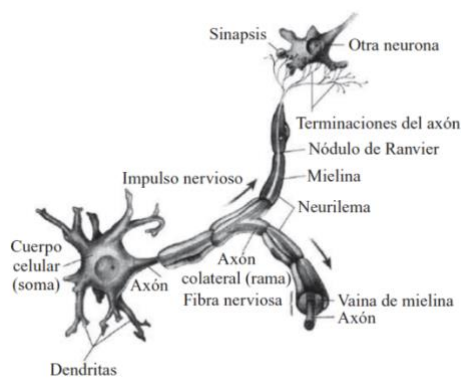
Es necesario comprender cómo se relaciona la neurona biológica con una neurona artificial en cuanto a su composición.

Una neurona biológica es una célula especializada en procesar información. Compuesta por el cuerpo de la célula (soma) y dos tipos de ramificaciones: axón y dendritas. La neurona recibe señales (impulsos) de otras neuronas mediante sus dendritas y transmite señales que se crean en el cuerpo mediante el axón. En la

Figura 1-5 se ilustra una neurona biológica y sus partes.

Las neuronas poseen un cuerpo esférico (más o menos) de 5 a 10 micras de diámetro, del que salen las dendritas y el axón. Se calcula que en el cerebro humano existen aproximadamente  $10^{15}$  conexiones entre neuronas.

La señal generada por la neurona y transportada por el axón es un impulso eléctrico, mientras que la señal entre axón-dendritas entre neuronas es de origen químico. Los vertebrados tienen un recubrimiento en sus axones con mielina que aumenta la velocidad de transmisión (120 m/s) comparado con los invertebrados (25 m/s).



Fuente: Inteligencia Artificial con Aplicaciones a la Ingeniería (Ponce Cruz, 2010).

Figura 1-5: Partes de una Neurona Biológica.

La similitud entre la actividad sináptica y la analogía con redes neuronales artificiales se debe a que las señales que llegan a la sinapsis son las entradas de la neurona; que son ponderadas a través de un parámetro llamado peso, que pueden excitar o inhibir a la neurona.

Luego se realiza la suma ponderada de las entradas. Si la suma es igual o mayor al umbral (matemático) de la neurona se activa y da salida. La habilidad de ajustar las señales es un mecanismo de aprendizaje.

1.7.2. Tipos de Redes Neuronales

Se diferencian en la manera de realizar las conexiones (Ponce Cruz, 2010):

- **Redes de propagación hacia delante (feed-forward):** El flujo de información siempre va hacia delante, extendiéndose por las capas, pero no existe ninguna conexión de retroalimentación.
- **Redes Recurrentes:** Existe conexión de retroalimentación, resultando en un proceso de evolución hacia un estado estable de la red.

1.7.3. Aplicaciones de las RNA

Las redes neuronales artificiales se aplican en diferentes industrias, en la Tabla 1-1 se muestran algunas de ellas y asociadas al mantenimiento industrial.

Tabla 1-1: Aplicaciones de las RNA.

| Área        | Aplicación                                                                                                                                   |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Automóviles | Sistemas de piloto automático. Detección de fallas por reconocimiento externo de vibraciones.                                                |
| Manufactura | Control de la producción y del proceso. Diagnóstico de fallas en el proceso y maquinarias. Inspección de calidad mediante sistemas visuales. |
| Transporte  | Diagnóstico de frenos en camiones. Sistemas de ruteo y seguimiento de flotas.                                                                |
| Electrónica | Control de procesos. Visión artificial. Análisis de Fallas.                                                                                  |

Fuente: Adaptación, Inteligencia Artificial con Aplicaciones a la Ingeniería (Ponce Cruz, 2010).

1.7.4. Ventajas y Desventajas de las RNA

En las redes neuronales artificiales existen algunos pro y contras en relación a su aplicación. En la Tabla 1-2 se muestran algunas de ellas.

Tabla 1-2:Ventajas y Desventajas de las RNA.

| Ventajas                                                                                                                                                    | Desventajas                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Las RNA son robustas, pueden fallar algunos elementos de procesamiento, pero la red continúa trabajando; que es lo contrario a la programación tradicional. | Se deben entrenar para cada problema. Es necesario realizar múltiples pruebas para determinar la arquitectura adecuada. El entrenamiento es largo y puede consumir varias horas de un computador. |
| Para utilizarla no es necesario conocer los detalles matemáticos. Sólo se requiere estar familiarizado con los datos.                                       | Debido a que las RNA se entrenan en lugar de programarlas, éstas necesitan muchos datos.                                                                                                          |
| La solución de los problemas no lineales es uno de los fuertes de las RNA.                                                                                  | Son métodos de “caja negra”, es imposible saber por qué el modelo de RNA alcanza un valor de salida de predicción.                                                                                |
| Pueden sintetizar algoritmos a través de un proceso de aprendizaje.                                                                                         |                                                                                                                                                                                                   |

Fuente: Inteligencia Artificial con Aplicaciones a la Ingeniería (Ponce Cruz, 2010).

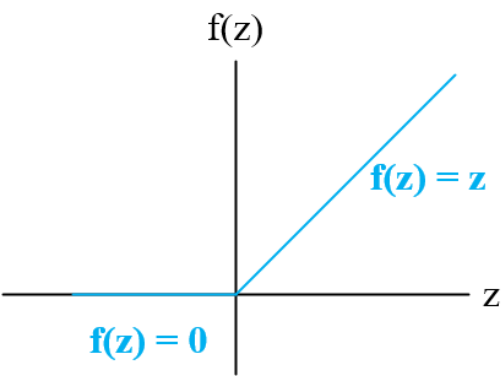
1.8. FUNCIONES DE ACTIVACIÓN

Generalmente se representa como  $g(z)$  o  $f(z)$ . Donde  $z$  es el resultado del calculo que realiza una neurona. Una función de activación devuelve un valor en la salida mediante la(s) entradas que reciba una neurona. Algunas de las funciones de activación utilizadas en una red neuronal se muestran a continuación.

1.8.1. ReLU (Rectified Logic Unit)

Se denota como en la ecuación (8). Tiene recorrido de  $[0,+\infty]$ . Es una de las más utilizadas en las capas ocultas y en problemas de regresión. Hace que el aprendizaje sea mucho más rápido. Forma parte de las funciones mixtas y se muestra en la Figura 1-6.

$$\begin{cases} 0 & : z \leq 0 \\ z & : z > 0 \end{cases} \tag{8}$$



Fuente: Elaboración Propia, basado en Inteligencia Artificial con Aplicaciones a la Ingeniería (Ponce Cruz, 2010).

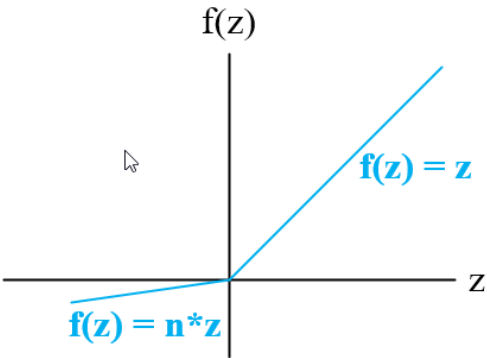
Figura 1-6: Función de Activación ReLU.

1.8.2. Leaky ReLU:

Está basado en ReLU, tiene una pequeña pendiente en el lado negativo de la función. Se denota como en la ecuación (9). Tiene recorrido de  $[-\infty,+\infty]$ . Con un factor  $n$  pequeño se evita que las neuronas “mueran”. Utilizada en problemas de regresión. Forma parte de las funciones mixtas y se muestra en la Figura 1-7.

$$\begin{cases} n * z & : z \leq 0 \\ z & : z > 0 \end{cases} \tag{9}$$

Donde:  
 $n$ : Número aleatorio.



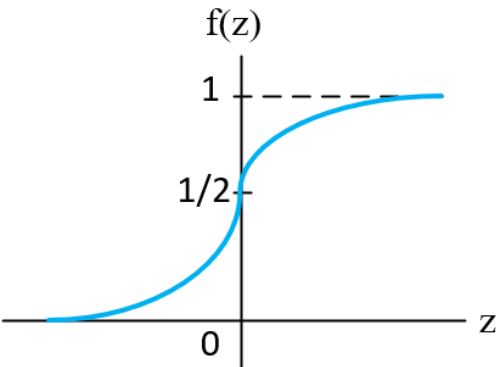
Fuente: Elaboración Propia, basado en Inteligencia Artificial con Aplicaciones a la Ingeniería (Ponce Cruz, 2010).

Figura 1-7: Función de Activación Leaky ReLU.

1.8.3. Sigmoide

Se denota como en la ecuación (10). Tiene recorrido de  $[0,1]$ . Usualmente se ocupa sólo en la capa de salida. Se emplea en problemas de clasificación debido a su recorrido, a menos que la salida esperada del modelo coincida con el recorrido. Se muestra en la Figura 1-8.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{10}$$



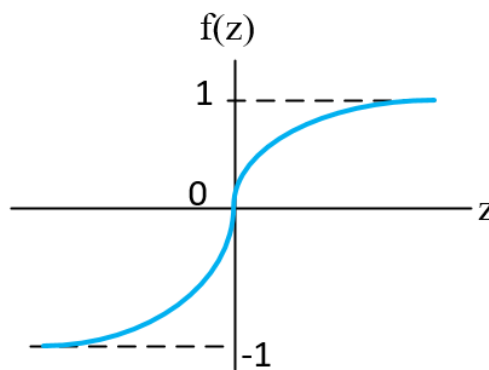
Fuente: Elaboración Propia, basado en Inteligencia Artificial con Aplicaciones a la Ingeniería (Ponce Cruz, 2010).

Figura 1-8: Función de Activación Sigmoide.

#### 1.8.4. Tangente Hiperbólica

Es similar a la función sigmoide en cuanto a su forma. Se denota como en la ecuación (11). Tiene recorrido de  $[-1,1]$ . También es una alternativa a sigmoide en las capas ocultas (no en salida). Se emplea en problemas de clasificación, a menos que la salida esperada coincida con el recorrido. Se muestra en la Figura 1-9.

$$\text{TanH}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (11)$$



Fuente: Elaboración Propia, basado en Inteligencia Artificial con Aplicaciones a la Ingeniería (Ponce Cruz, 2010).

Figura 1-9: Función de Activación Tangente Hiperbólica.

### 1.9. ENTRENAMIENTO DE REDES NEURONALES

Es el proceso de configuración de una red neuronal para que las entradas produzcan las salidas deseadas a través del fortalecimiento de las conexiones. Se pueden establecer si se conocen los pesos previamente o con técnicas de retroalimentación y aprendizaje que cambian los pesos hasta hallar los más adecuados.

Las formas de aprendizaje pueden ser:

- **Supervisado o Asociativo:** Se introducen las entradas que corresponden a determinadas salidas.
- **No Supervisado o Autoorganizado:** Se enfoca en encontrar características estadísticas entre agrupamientos de patrones de entradas.

### 1.9.1. Iniciación de los parámetros

Tal como en la ecuación (12) los pesos “w” y sesgos “b” deben inicializarse con:

- **Valores Aleatorios:** No con ceros. Esto evita que todas las neuronas de la misma capa aprendan la misma función.
- **Valores próximos a cero:** Agiliza y mejora el aprendizaje:

$$w = rnd * 0,01 \ ; \ b = rnd * 0,01 \quad (12)$$

Para que la red esté lista para entrenar, se necesita tomar tres cosas como parte del paso de compilación (Chollet, 2018):

- **Métricas para monitorear el entrenamiento y prueba:** Procurando obtener una buena precisión del modelo.
- **Función de Pérdida:** Para que la red sea capaz de medir su desempeño con los datos de entrenamiento, para poder dirigirse al camino correcto.
- **Optimizador:** Mecanismo mediante la red se actualizará a sí misma basándose en los datos de entrada y función de pérdida.

Dado las características del presente trabajo de título, enfocado en problemas de regresión, la función de pérdida puede coincidir con alguna métrica a utilizar.

### 1.9.2. Retro propagación del Error (Back propagation)

Llamado también como diferenciación en reversa. Es un algoritmo que permite entrenar redes neuronales, de forma que en base al error que tiene el modelo, corrige gradualmente ese error hasta minimizar el error. Cada neurona minimiza su función de costo o pérdida modificando sus pesos.

### 1.9.3. Descenso de Gradiente Estocástico

El gradiente de una función es la derivada en la que la pendiente es el óptimo. Se utiliza para actualizar la función que representa una red neuronal. El objetivo es minimizar la función de pérdida. La tasa de aprendizaje permite realizar el descenso de manera gradual hasta encontrar el valor mínimo de función de pérdida.

Ya que inicialmente las matrices de pesos y sesgo se llenan con números aleatorios. La idea es ajustar los valores gradualmente, en base a una señal de retroalimentación. A continuación se resume el proceso iterativo de entrenamiento:

1. Seleccionar una muestra de los datos de entrada para entrenamiento y sus correspondientes valores esperados de salida.
2. Hacer funcionar la red con los valores de entrada para encontrar los valores de predicción (proceso feed forward).
3. Calcular la perdida de la red en la muestra, midiendo entre los valores esperados y de predicción.
4. Calcular el gradiente de la perdida respecto a los parámetros de la red (back propagation).
5. Mover un poco los parámetros en dirección opuesta desde el gradiente, para reducir la pérdida gradualmente.

1.9.4. Momentum

Si la tasa de aprendizaje es demasiado pequeña, el proceso de optimización se podría atascar en un valor mínimo local en vez del mínimo global. Se puede evitar este problema utilizando “momentum”, inspirado en la física.

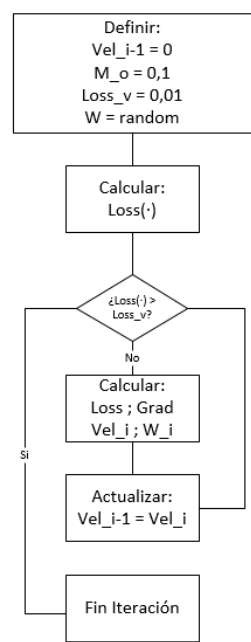
Este método que complementa el optimizador del descenso de gradiente se expresa mediante la ecuación (13). Y el valor de los pesos se actualiza mediante la ecuación (14).

$$Vel_i = Vel_{i-1} * M_o + \eta * Grad \tag{13}$$

$$w_i = w_{i-1} + M_o * Vel - \eta * Grad \tag{14}$$

Donde:  
 $Vel_i$ : "Velocidad".  
 $M_o$ : Momentum.  
 $Grad$ : Gradiente.  
 $\eta$ : Tasa de Aprendizaje.

Utilizando las ecuaciones anteriores, se resume el proceso de momentum ilustrado en la Figura 1-10.



Fuente: Elaboración Propia, basado en Deep Learning with Python (Chollet, 2018).

Figura 1-10: Proceso de Momentum.

1.9.5. Regla de la Cadena

Matemáticamente, si  $y = f(u)$  ;  $u = g(x)$  ;  $y = f(g(x))$  entonces la derivada resulta como en la ecuación (15).

$$\frac{dy}{dx} = \frac{dy}{du} * \frac{du}{dx} \tag{15}$$

Para realizar la retro propagación del error, se necesita calcular una serie de derivadas (gradiente), siguiendo la ecuación (15). Para los pesos se utiliza como referencia la ecuación (16). Mientras que, para actualizar los valores de sesgo, la ecuación (17) es la adecuada.

$$\frac{\partial L(\hat{y}, y)}{\partial w_n} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial w_n} \tag{16}$$

$$\frac{\partial L(\hat{y}, y)}{\partial b} = \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial b} \tag{17}$$

Un ejemplo de cómo hallar los valores de la derivada con una función de activación sigmoide, se encuentra en el Anexo C.

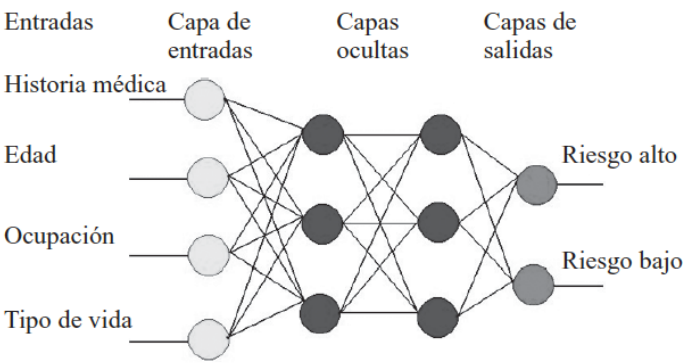
Como se explicó con el optimizador de momentum, para contrastar se tiene el descenso de gradiente estocástico. Entonces, la actualización de los valores de pesos y sesgo para una capa “l”, está dado por las ecuaciones (18) y (19).

$$w^{[l]} \leftarrow w^{[l]} - \eta * \frac{1}{n} \sum_{i=1}^m \frac{\partial L(\hat{y}, y)}{\partial w^{[l]}} \tag{18}$$

$$b^{[l]} \leftarrow b^{[l]} - \eta * \frac{1}{n} \sum_{i=1}^m \frac{\partial L(\hat{y}, y)}{\partial b^{[l]}} \tag{19}$$

**1.10. ESTRUCTURA DE REDES NEURONALES**

Una red neuronal puede tener una estructura arbitraria. Tal como en la Figura 1-11 o Figura 1-4. A las entradas externas son aplicadas en la primera capa, y las salidas se consideran la última capa. Por convención, las capas internas que no se observan se denominan capas ocultas.



Fuente: Inteligencia Artificial con Aplicaciones a la Ingeniería (Ponce Cruz, 2010).

Figura 1-11: Estructura RNA Feed-forward en problemas de clasificación.

A continuación se mencionan algunas de las redes neuronales tipo feed forward comunes para comprender su estructura.

### 1.10.1. Perceptrón

Puede entenderse como la neurona artificial y unidad básica que realiza la operación de una suma ponderada de las entradas externas aplicadas a éste. Los pesos pueden ser positivos o negativo.

La red tipo perceptrón fue inventada por el psicólogo Frank Rosenblatt en el año 1957. Su intención era ilustrar algunas propiedades fundamentales de los sistemas inteligentes en general, sin entrar en mayores detalles con respecto a condiciones específicas y desconocidas para organismos biológicos concretos.

El entrenamiento de un perceptrón es por medio de la regla de aprendizaje delta. Para cada peso  $\omega$  se realiza un ajuste según la ecuación (20).

$$d\omega = \eta(y - Y)X \quad (20)$$

Donde:

$\eta$ : Tasa de aprendizaje.

$y$ : Valor deseado.

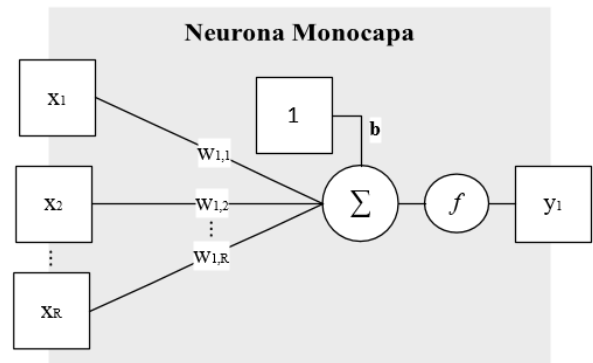
$Y$ : Valor obtenido.

$X$ : Entrada aplicada al perceptrón.

La tasa de aprendizaje es una constante positiva pequeña, con valores entre 0 y 1. Además, una  $\eta$  efectiva puede inducir el proceso de mejora de la exactitud de resolución (clasificación o regresión).

### 1.10.2. Neurona Monocapa

Se diferencia del perceptrón ya que tiene un número de entradas mayor a dos. Esta red se representa como en la Figura 1-12.



Fuente: Elaboración Propia, basado en la estructura de una RNA.

Figura 1-12: Ejemplo de Neurona Monocapa.

La notación matricial de la neurona monocapa está dada por la ecuación (21). Resultando en una matriz de 1 fila y 1 columna.

$$W_{1 \times 1} \times X_{R \times 1}^T + b_{1 \times 1} = Z_{1 \times 1} \tag{21}$$

- En donde:
- W: matriz de pesos.
- X: matriz de entradas.
- b: matriz de sesgo.
- Z: matriz resultante.

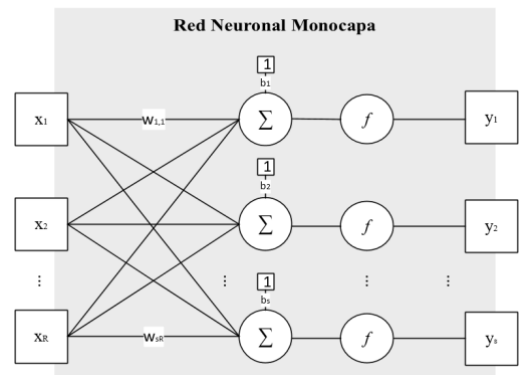
Las matrices quedan como en la ecuación (22) representando la suma ponderada, lista para utilizar en alguna función de activación.

$$\begin{aligned} [w_{1,1} \quad \dots \quad w_{1,R}]_{1 \times R} \times \begin{bmatrix} x_1 \\ \vdots \\ x_R \end{bmatrix}_{R \times 1} + [b]_{1 \times 1} \\ = [w_{1,1} * x_1 + \dots + w_{1,R} * x_R + b]_{1 \times 1} \end{aligned} \tag{22}$$

- En donde:
- w: pesos.
- x: entradas de la red.
- R: total de entradas.
- b: sesgo.

1.10.3. Red Neuronal Monocapa

En esta red se introducen más neuronas en una misma capa. Cada entrada se conecta a todas las neuronas. Se representa como en la Figura 1-13.



Fuente: Elaboración Propia, basado en la estructura de una RNA.

Figura 1-13: Ejemplo de Red Neuronal Monocapa.

La notación matricial de la red neuronal monocapa está dada por la ecuación (23). Resultando en una matriz de “s” filas y 1 columna.

$$W_{sxR} \times X_{Rx1}^T + b_{sx1} = Z_{sx1} \tag{23}$$

Las matrices quedan como en la ecuación (24) representando la suma ponderada, lista para utilizar en alguna función de activación. La matriz de pesos contiene “s” filas idénticas, mientras que a la matriz de sesgo contiene “s” filas, un valor de sesgo por cada neurona de la red.

$$\begin{aligned} &\begin{bmatrix} w_{1,1}^{[l]} & \dots & w_{1,R}^{[l]} \\ \vdots & \ddots & \vdots \\ w_{s,1}^{[l]} & \dots & w_{s,R}^{[l]} \end{bmatrix}_{sxR} \times \begin{bmatrix} x1 \\ \vdots \\ xR \end{bmatrix}_{Rx1} + \begin{bmatrix} b1 \\ \vdots \\ bs \end{bmatrix}_{sx1} \\ &= \begin{bmatrix} w_{1,1} * x_1 + \dots + w_{1,R} * x_R + b_1 \\ \vdots \\ w_{s,1} * x_1 + \dots + w_{s,R} * x_R + b_s \end{bmatrix}_{sx1} \end{aligned} \tag{24}$$

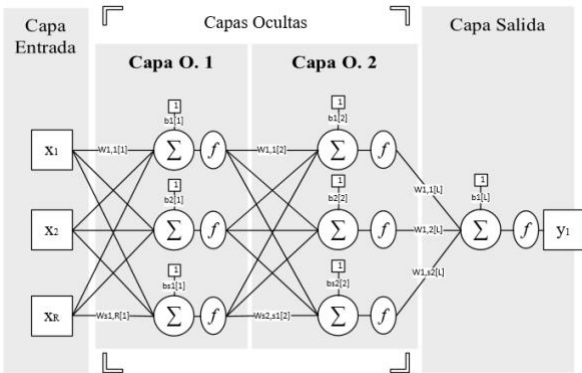
- En donde:
- w: pesos.
  - x: entradas de la red.

- R: total de entradas.
- b: sesgo.
- s: total de neuronas.

Existen leves diferencias en la capa de salida y funciones de activación cuando se requiere de una RNA para problemas de clasificación o regresión.

1.10.4. MLP para Regresión

Basado en los anteriores puntos, esta arquitectura se compone de varias redes monocapa conectadas de forma secuencial con una neurona monocapa al final de ésta, tal como se muestra en la Figura 1-14.



Fuente: Elaboración Propia, basado en la estructura de una RNA.

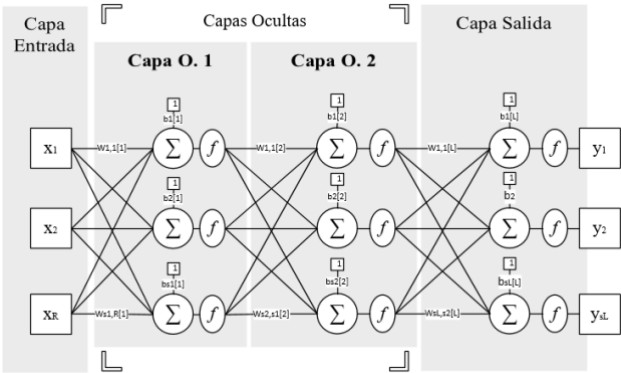
Figura 1-14: Perceptrón Multicapa para Problemas de Regresión.

En forma matricial la red se escribe como en la ecuación (25), de lo que se obtiene una matriz de salida de 1 fila y 1 columna.

$$y_{1 \times 1} = f^{[L]} \left( W_{1 \times 1}^{[L]} \times f^{[2]} \left( W_{s \times R}^{[2]} \times f^{[1]} \left( W_{s \times R}^{[1]} \times X_{R \times 1}^{T[1]} + b_{s \times 1}^{[1]} \right) + b_{s \times 1}^{[2]} \right) + b_{1 \times 1}^{[L]} \right) \tag{25}$$

1.10.5. MLP para Clasificación

En una red neuronal feed-forward para clasificación, las capas de entradas y capas ocultas mantienen su forma. Mientras que en la capa de salida se utiliza una red neuronal monocapa, tal como en la Figura 1-15.



Fuente: Elaboración Propia, basado en la estructura de una RNA.

Figura 1-15: Perceptrón Multicapa para Problemas de Clasificación.

La denotación matricial de la red está dada por la ecuación (26) que representa su estructura, de lo que se obtiene una matriz de salida de “sL” cantidad de filas y 1 columna.

$$y_{s_L x_1} = f^{[L]} \left( W_{sxR}^{[L]} \times f^{[2]} \left( W_{sxR}^{[2]} \times f^{[1]} \left( W_{sxR}^{[1]} \times X_{Rx1}^{T[1]} + b_{sx1}^{[1]} \right) + b_{sx1}^{[2]} \right) + b_{sx1}^{[L]} \right) \tag{26}$$

## **CAPÍTULO 2: FORMULACIÓN DE UN MODELO DE ML**

2. FORMULACIÓN DE UN MODELO DE ML

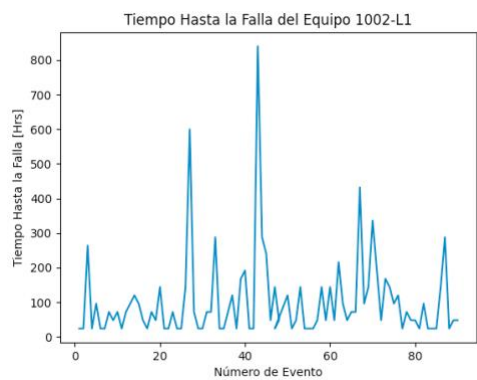
Actualmente se cuenta con una base de datos de las fallas mecánicas de la línea 1 de celulosa Arauco del equipo con el tag 1002 (Debarking Drum). Se realiza un filtrado en el cual se censuran eventos de Mantenición programada, Inspecciones y se consideran tiempos fuera de servicio asociado al área mecánica.

La base datos matriz analizado corresponde de las fechas desde enero 2016 hasta el 22 enero 2019. Por lo que se guarda en formato Excel para importar hacia Python y procesar los datos.

2.1. CONJUNTO DE DATOS

Los datos obtenidos desde la base de datos matriz corresponde a los eventos ocurridos entre las fechas: Enero 2016 al Agosto 2018.

El conjunto de datos importado a Python contiene causa de falla, equipo correspondiente, tiempo fuera de servicio, fecha del evento, tiempo hasta la falla y número de evento. En la Figura 2-1 se puede observar que a medida que por las acciones de mantenimiento realizadas al equipo, el tiempo hasta la falla incrementa en función del tiempo.



Fuente: Elaboración Propia con Python.

Figura 2-1: Gráfico Tiempos Hasta la Falla.

La modo de la falla en el equipo, según la información de la base de datos, corresponde a detenciones del sistema de lubricación, ocasionando que operaciones tenga que lubricar manualmente ciertos puntos del equipo.

## 2.2. MODELO ML CONTRA DISTRIBUCIÓN WEIBULL

Se utiliza un modelo basado en (Baptista, Sankararaman, de Medeiros, Nascimento Jr., & Henriquesa, 2018) en el cual se proponen ciertos cambios.

Para lograr la predicción de eventos de falla, se utilizan los siguientes datos:

- Distribución Weibull, donde  $X \sim Weibull(\alpha, \beta)$ :
  - Función de Densidad de Probabilidad de Fallas, denotado como  $f(t)$ .
  - Función de Distribución Acumulada, denotado como  $F(t)$ .
  - Función de Confiabilidad, denotado como  $R(t)$ .
  - Función de Tasa de Fallas, denotado como  $\lambda(t)$ .
- Modelo ARMA: Modelo Previo para Pronóstico de Eventos de Fallas.
- ARIMA+RNA: Modelo Propuesto para los Datos de Eventos de Falla.

Esto significa que primero se debe modelar la confiabilidad del equipo para estimar los parámetros de la distribución Weibull ( $\alpha$  y  $\beta$ ) para para evaluar su precisión. Luego entrenar el modelo predictivo propuesto para comparar los resultados obtenidos.

### 2.2.1. Confiabilidad con Distribución Weibull

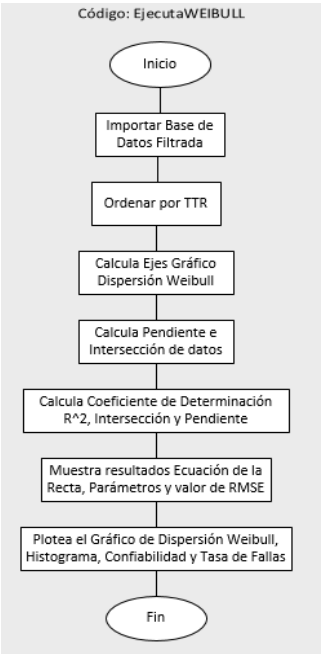
El código utilizado se encuentra detallado en el Anexo D, y se encuentra ilustrado el flujo de trabajo en la Figura 2-2. Muestra el proceso de cálculo de los parámetros de Weibull mediante mínimos cuadrados.

Además se calcula el coeficiente de determinación que no es mas que el cuadrado del coeficiente de correlación de Pearson, tal como en la ecuación (27). El código imprime en pantalla el resultado de la ecuación de la recta de la regresión lineal hecha en el gráfico de dispersión de Weibull. También se imprime el resultado de la métrica de precisión RMSE.

El código elaborado es funcional y simple, lo que facilita el cálculo y visualización de información respecto a la modelación de confiabilidad mediante Python.

$$R^2 = r_{xy}^2 = \left( \frac{\sigma_{xy}}{\sigma_x * \sigma_y} \right)^2$$

(27)

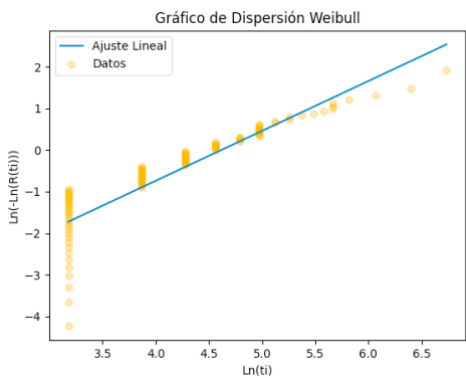


Fuente: Elaboración Propia con Visio.

Figura 2-2: Diagrama de Flujo del código “EjecutaWEIBULL”.

El gráfico de dispersión de Weibull es una prueba de bondad de ajuste en relación a los parámetros estimados. La ecuaciones utilizadas son la función de densidad de fallas, la ecuación de función de fallas acumuladas, la ecuación de confiabilidad y la esperanza o tiempo medio entre fallas corresponde.

Se obtienen los resultados en la Tabla 2-1 correspondientes a la estimación de parámetros de la distribución Weibull que se muestran en la Figura 2-3.



Fuente: Elaboración Propia con Python.

Figura 2-3: Gráfico de Dispersión Weibull.

Tabla 2-1: Resultados Estimación Parámetros Weibull.

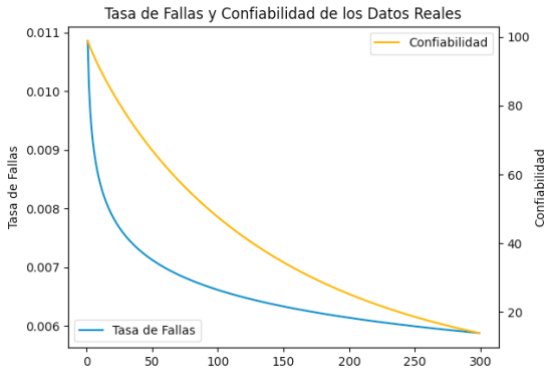
| Concepto                | Valor             |
|-------------------------|-------------------|
| Alpha                   | 139,9             |
| Beta                    | 0,89              |
| Coef. Determinación R^2 | 0,739             |
| Ecuación de la Recta    | $0,89 * X - 4,41$ |
| MTBF                    | 147,9             |

Fuente: Elaboración Propia, basado en resultados del código utilizado.

La modelación de la confiabilidad mediante Python indica que el equipo tiene un Alpha de 139,9 [horas/falla], que es el parámetro de escala o vida característica, lo que corresponde al 36,79% de la confiabilidad o un 63,21% de probabilidad de fallas acumuladas. Mientras que al indicar un valor de beta de 0,89 se puede decir que el equipo se encuentra en etapa de mortalidad infantil.

El coeficiente de determinación junto con el grafico de dispersión de la Figura 2-3 es una prueba de bondad de ajuste de los parámetros obtenidos. Se obtiene un MTBF de 147,9 [horas/falla] corresponde al 34,96% de confiabilidad del equipo o un 65,04% de probabilidad de fallas acumuladas. Se espera que en promedio (inferencia estadística) el equipo falle a esa cantidad de horas.

En la Figura 2-4 se puede observar la confiabilidad y tasa de fallas correspondientes, la tasa de fallas aumenta con el tiempo, mientras que la confiabilidad siempre disminuye con el paso del tiempo.

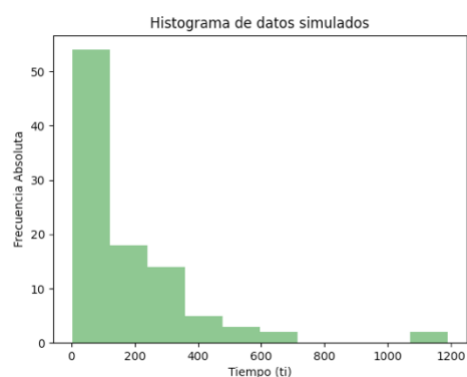


Fuente: Elaboración Propia con Python.

Figura 2-4: Tasa de Fallas y Confiabilidad de los Datos Reales.

Se crean eventos aleatorios para medir la métrica de RMSE, comparando los eventos aleatorios contra la base de datos filtrada. Es por esto que se simulan 98 eventos aleatorios para continuar con el análisis. Por lo tanto, en la Figura 2-5 se puede ver la distribución de los datos con bin o intervalo de 10.

La forma observada puede asociarse aproximadamente a una curva Exponencial Negativa (la cual no está validada en el presente trabajo) como función de densidad de fallas. Lo bueno de la distribución Weibull es que es versátil y puede cambiar su forma mediante su parámetro de forma  $\beta$ ).



Fuente: Elaboración Propia con Python.

Figura 2-5: Histograma de los Datos Simulados en Python.

Finalmente, el resultado obtenido de la métrica RMSE es de 673,2 para el conjunto de datos de validación. Se utiliza como punto inicial para comparar los demás modelos del presente trabajo.

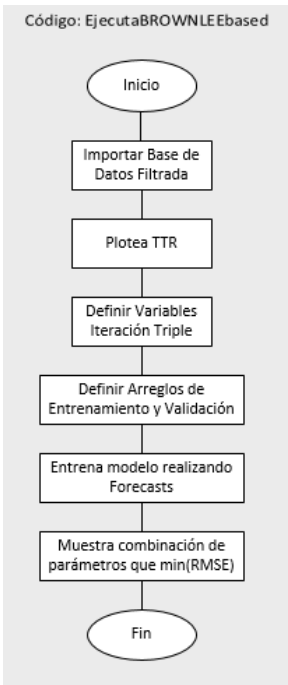
**2.3.      MODELO PREDICTIVO DE ML: ARIMA + RNA:**

El modelo predictivo consta de dos partes, donde la primera encontrar los parámetros óptimos para un modelo ARIMA, seguido del entrenamiento de una red neuronal multicapa para regresión tipo feed-forward.

2.3.1. Código de Modelo ARIMA

El código utilizado se encuentra detallado en el Anexo D, y se encuentra ilustrado el flujo de trabajo en la Figura 2-6. Muestra el proceso de cálculo de los parámetros del modelo ARIMA mediante una iteración triple con el objetivo de minimizar la métrica de precisión RMSE. Cabe mencionar que para coincidir con (Baptista, Sankararaman, de Medeiros, Nascimento Jr., & Henriquesa, 2018) se utiliza un modelo ARIMA con parámetro de integración  $d = 0$ .

Se ejecuta un modelo ARIMA definiendo el arreglo con valores de eventos de fallas impares para el entrenamiento. Mientras que los valores pares corresponden al arreglo de datos para validación.



Fuente: Elaboración Propia con Visio.

Figura 2-6 : Diagrama de Flujo del código “EjecutaBROWNLEEbased”.

Se determina que el modelo ARIMA de entrenamiento que minimiza la métrica RMSE con 125,8 para entrenamiento y 137,1 para validación, es ARIMA(1,0,0) o su equivalente AR(1). Las predicciones del modelo en el conjunto de datos de validación del modelo se encuentran en la Figura 2-8.

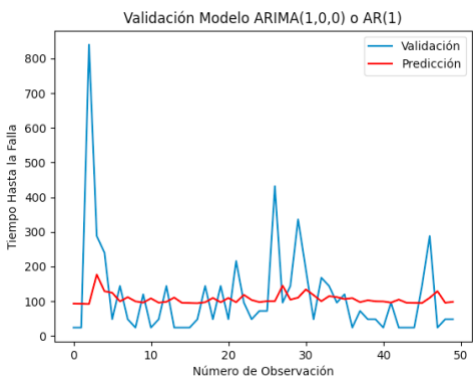
Una vez encontrado el parámetro del modelo ARIMA, se procede a ejecutar el código “EjecutaARMA1” el cual se detalla en el Anexo F. Se ilustra el flujo de trabajo del código en la Figura 2-7.



Fuente: Elaboración Propia con Visio.

Figura 2-7: Diagrama de Flujo del código “EjecutaARMA1”.

Se puede observar que durante el conjunto de datos de validación de la Figura 2-8 el modelo AR(1) tiende a predecir eventos de fallas por un valor promedio sobre el tiempo en el cual ocurren eventos. En rojo se ven las predicciones del modelo y en azul se encuentran los datos reales del equipo (validación).



Fuente: Elaboración Propia con Python.

Figura 2-8: Conjunto de Validación del Modelo ARIMA (1,0,0).

Se determina que los valores de predicción del modelo para el conjunto de validación en su mayoría logra pronósticos que están por sobre el tiempo del evento de falla verdadero.

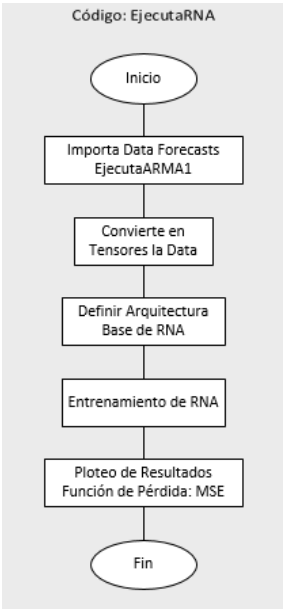
A pesar de que el modelo AR(1) logra disminuir la métrica de precisión respecto a Weibull, no es lo suficientemente bueno como para predecir antes de que ocurran los eventos de falla.

2.3.2. Código de Modelo de RNA

En Python se utiliza librerías como keras y tensorflow, mediante el cual se simplifica la programación cuando se importan. El código utilizado se encuentra detallado en el Anexo G y el flujo de trabajo se ilustra en la Figura 2-9Figura 2-9.

Algunos de los conceptos clave para este modelo son:

- 1. **Epochs o Épocas:** Veces en la que todos los datos de entrenamiento pasan por la red neuronal, en donde: feed-forward + backpropagation = 1 epoch.
- 2. **Batch Size o Tamaño de Lote:** Consiste en particionar los datos de entrenamiento/validación en una cantidad más pequeña para actualizar el gradiente, el tamaño dependerá de la memoria del computador sobretodo si la data es de gran volumen.



Fuente: Elaboración Propia con Visio.

Figura 2-9 : Diagrama de Flujo del código “EjecutaRNA”.

Se define arbitrariamente la arquitectura de la red neuronal, resultando en 2 entradas: la primera con los valores de la predicción del modelo AR(1) y la segunda con los valores al cuadrado, generando un aumento de la dimensión de la variable. Se decide por 5 capas ocultas, en la cual cada una contiene 10 neuronas. Y finalmente 1 capa de salida, ya que es un problema de regresión.

En todas las neuronas de las capas ocultas se utilizan funciones de activación LeakyReLU (excepto en la salida) de parámetro  $\alpha = 0,001$  para evitar que las neuronas “mueran” durante el entrenamiento como puede pasar en ReLU.

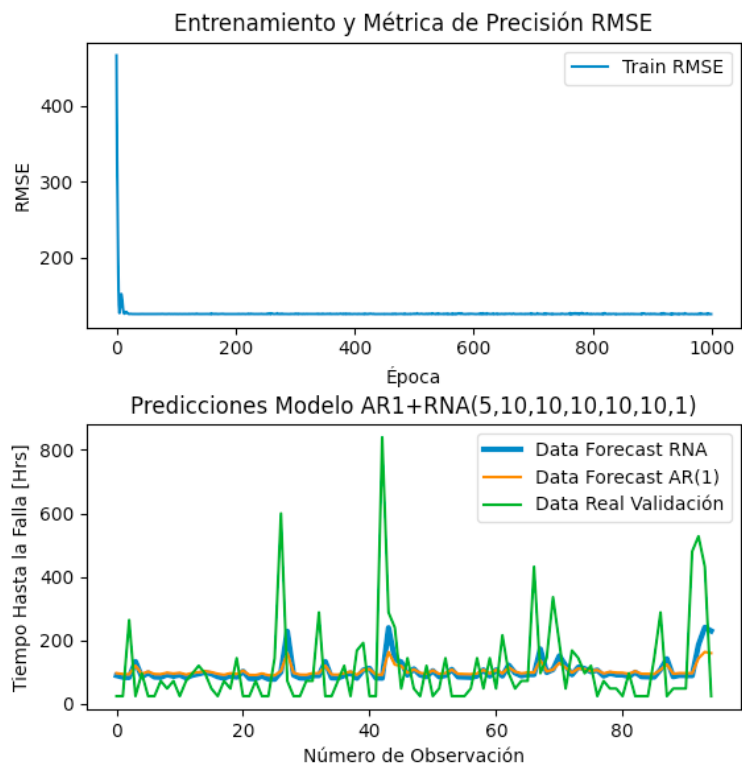
Se considera una función de pérdida MSE o Error Cuadrático Medio y una métrica de precisión RMSE o Raíz del Error Cuadrático Medio. El número de épocas definidas es de 1000. Al igual que en el modelo ARIMA entrenado, se define como conjunto de entrenamiento todos los eventos menos los 5 últimos para entrenamiento y se valida el modelo con todos los datos.

### 2.3.3. Resultados Obtenidos del Modelo Base

En el código expuesto anteriormente, el modelo base corresponde a un RNA(2,10,10,10,10,10,1) que considera como entrada del modelo los pronósticos del modelo AR(1) logrando establecer una conexión en serie entre los dos modelos.

Se obtiene un total de 481 parámetros del modelo, de sesgo y pesos. La capa de entrada contiene los primeros 30 parámetros. Las capas ocultas, de la 1 a 4 contienen 110 parámetros por capa. Mientras que la ultima capa oculta contiene los últimos 11 parámetros.

En la Figura 2-10 se ve el resultados del comportamiento de la métrica de precisión a través de las épocas del entrenamiento del modelo. El valor del RMSE converge en 125,4 para el entrenamiento y al evaluar el modelo se obtiene un RMSE de 136,2 para validación.

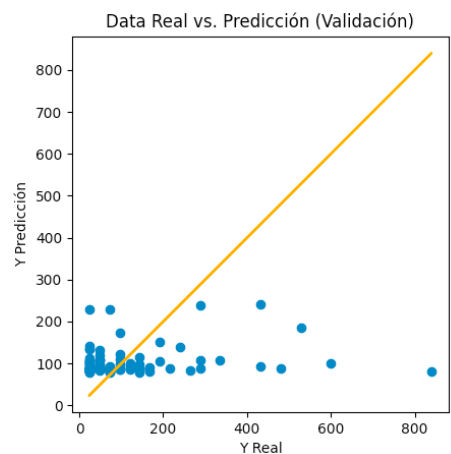


Fuente: Elaboración propia con Python.

Figura 2-10: Validación del Modelo AR(1)+RNA(5,10,10,10,10,10,1).

Con el modelo se logran predicciones del tiempo hasta la falla con valores siempre mayores a cero, se puede ver que en eventos de corto plazo el modelo si predice el tiempo de hasta la falla. Lamentablemente, no son resultados que permitan una anticipación temprana a los eventos de falla observados en el equipo para eventos de mediano y largo plazo, tal como se ilustra en la Figura 2-10.

En la Figura 2-11 se puede ver que el modelo no logra realizar una predicción precisa de los eventos de fallas. Lo que se debería ver en la figura es que los todos los puntos del grafico se encuentren lo mas cerca a la línea color naranja.



Fuente: Elaboración Propia con Python.

Figura 2-11: Data Real vs. Predicción en Modelo Base.

Como se ve en la Tabla 2-2, el modelo base propuesto solamente llega a superar a la distribución Weibull. Mientras que el modelo AR(1) es mejor, en cuanto a la métrica de precisión de modelo RMSE.

Además, con el modelo final se pretende minimizar los resultados parciales hasta el momento, para lograr una predicción temprana del tiempo de eventos de fallas observados en el equipo, idealmente antes de que estas ocurran.

Tabla 2-2: Resultados RMSE.

| Modelo                          | RMSE Entrenamiento | RMSE Validación |
|---------------------------------|--------------------|-----------------|
| Distribución Weibull            | N/A                | 673,2           |
| AR(1)                           | 125,8              | 137,1           |
| AR(1) + RNA(2,10,10,10,10,10,1) | 125,4              | 136,2           |

Fuente: Elaboración propia con Datos de Python.

Los principales aportes del presente capítulo fueron lograr el funcionamiento de la herramienta de machine learning Redes Neuronales Artificiales en serie con un modelo AR(1), en un problema de regresión para pronosticar eventos de fallas.

Los retos del próximo capítulo consiste en poder optimizar los resultados iniciales, de manera que la métrica se minimice. Lo que significa cambiar las variables de la arquitectura de la red neuronal (capa de entrada, capas ocultas y número de neuronas por capa oculta), épocas para ir iterando hasta obtener un resultado final.

## **CAPÍTULO 3: EVALUACIÓN DE UN MODELO ÓPTIMO DE ML**

### 3. EVALUACIÓN DE UN MODELO ÓPTIMO DE ML

El modelo final tiene que cumplir con reducción de la métrica de precisión RMSE y lograr predecir idealmente antes de que ocurran las fallas. Es por esto que a continuación se realiza un proceso iterativo.

#### 3.1. CAMBIOS EN EL MODELO BASE

Para lograr un mejor desempeño del modelo base propuesto anteriormente, se comienza con una iteración sobre las variables de entradas del modelo, cantidad de capas ocultas y neuronas por capa, dejando intacto todo lo demás.

##### 3.1.1. Iteración sobre Modelo Base

Se itera sobre el número de entradas, número de capas ocultas y número de neuronas por capa. En la Tabla 3-1 se muestra la codificación de las entradas para la iteración de modelos. Se conserva el número de épocas en 1.000 y la función de activación LeakyReLU como en el modelo base.

Se establece una codificación tipo binaria para poder identificar de forma ágil las entradas al momento de iterar. Además, se decide por eliminar la entrada de predicción  $AR(1)^2$  porque hace mas lento el proceso de aprendizaje.

##### 3.1.2. Procedimiento de Iteración

Mediante el código “EjecutaRNA.py” se obtiene solamente un modelo, por lo que hay que cambiar los parámetros según el modelo que se requiere entrenar.

Esta parte del trabajo es la que mas toma tiempo, y para evitar errores de digitación se decide por realizar el proceso iterativo de la siguiente forma:

1. Seleccionar la cantidad de capas que tendría la red neuronal artificial a utilizar, llamando alguna de las funciones: “Red\_1capa”, “Red\_2 capas”, “Red\_3capas”, “Red 4 capas”.
2. Hacer coincidir los parámetros de entrada, seleccionando el tipo de entrada mediante los arreglos de datos llamados: “X\_train”, “Y\_train”, “X\_test”,

- “Y\_test”. Se deben seleccionar los índices necesarios para que correspondan a los títulos de la base de datos importada.
- Ya que el código permite guardar el modelo para su posterior aplicación tras terminar su validación, es necesario definir un nombre por ejemplo: “Modelo 10-5”. Donde se generará un archivo en formato “tf” que es de la librería tensorflow.
  - Posterior a eso, se debe cargar el modelo y ejecutar los pronósticos de eventos de fallas ingresando las entradas correspondientes del modelo propuesto.

Tabla 3-1: Primera iteración sobre las entradas del modelo.

| Modelo | AR(1) | f(t) | R(t) | TF(t) |
|--------|-------|------|------|-------|
| 0      | 0     | 0    | 0    | 0     |
| 1      | 1     | 0    | 0    | 0     |
| 2      | 0     | 1    | 0    | 0     |
| 3      | 1     | 1    | 0    | 0     |
| 4      | 0     | 0    | 1    | 0     |
| 5      | 1     | 0    | 1    | 0     |
| 6      | 0     | 1    | 1    | 0     |
| 7      | 1     | 1    | 1    | 0     |
| 8      | 0     | 0    | 0    | 1     |
| 9      | 1     | 0    | 0    | 1     |
| 10     | 0     | 1    | 0    | 1     |
| 11     | 1     | 1    | 0    | 1     |
| 12     | 0     | 0    | 1    | 1     |
| 13     | 1     | 0    | 1    | 1     |
| 14     | 0     | 1    | 1    | 1     |
| 15     | 1     | 1    | 1    | 1     |

Fuente: Elaboración Propia con Excel.

3.1.3. Formato de Planilla de Iteración

La estrategia seleccionada es que arbitrariamente se decide que el número de entradas podrá ser 1, 2, 3 o 4. Lo que resulta en  $2^4 = 16$  modelos iniciales posibles. Se ignora la primera, ya que según la Tabla 3-1, el modelo 0 no tiene ninguna entrada.

Mientras que la cantidad de neuronas por capa podrá ser de 8, 16, 32 o 64. Lo que resulta en  $4^k$ , donde k es la cantidad de entradas por modelo, lo que es igual a 624 combinaciones totales posibles. Lo anterior se iguala al número de capas. En la Tabla 3-2 se muestra el formato utilizado para realizar cada una de las iteraciones

La idea es correr el código “EjecutaRNA.py” 624 veces para recolectar información y decidir por el mejor modelo de predicción de eventos de fallas. En la planilla se detalla el número de neuronas por capa, el número de parámetros por modelo, el RMSE de entrenamiento, RMSE de validación. Además, se escribe si algún modelo supera el modelo inicial AR(1), como también señalar si el modelo logra realizar una predicción de eventos de fallas ya que el código permite visualizar los resultados.

Tabla 3-2: Formato de Planilla de Iteración sobre Modelos.

| Modelo | Entradas | Codigo | Mod | RNA              | 1  | 2 | 3 | 4 | Parámetros | RMSE_t | RMSE_v | Mejora AR(1) | Anticipación |
|--------|----------|--------|-----|------------------|----|---|---|---|------------|--------|--------|--------------|--------------|
| 1      | 1        | 1000   | 1.1 | (1,8,0,0,0,0,1)  | 8  | 0 | 0 | 0 | 25         | 124,1  | 136,6  | Si           | No           |
|        |          |        | 1.2 | (1,16,0,0,0,0,1) | 16 | 0 | 0 | 0 | 49         | 124,1  | 136,7  | Si           | No           |
|        |          |        | 1.3 | (1,32,0,0,0,0,1) | 32 | 0 | 0 | 0 | 97         | 124,1  | 136,7  | Si           | No           |
|        |          |        | 1.4 | (1,64,0,0,0,0,1) | 64 | 0 | 0 | 0 | 193        | 124,1  | 136,7  | Si           | No           |
| 2      | 1        | 0100   | 2.1 | (1,8,0,0,0,0,1)  | 8  | 0 | 0 | 0 | 25         | 128,3  | 144,7  | No           | No           |
|        |          |        | 2.2 | (1,16,0,0,0,0,1) | 16 | 0 | 0 | 0 | 49         | 125,6  | 141,3  | No           | No           |
|        |          |        | 2.3 | (1,32,0,0,0,0,1) | 32 | 0 | 0 | 0 | 97         | 124,3  | 138,9  | No           | No           |
|        |          |        | 2.4 | (1,64,0,0,0,0,1) | 64 | 0 | 0 | 0 | 193        | 124,3  | 138,8  | No           | No           |
| 4      | 1        | 0010   | 4.1 | (1,8,0,0,0,0,1)  | 8  | 0 | 0 | 0 | 25         | 136,5  | 153,8  | No           | No           |
|        |          |        | 4.2 | (1,16,0,0,0,0,1) | 16 | 0 | 0 | 0 | 49         | 132    | 148,9  | No           | No           |
|        |          |        | 4.3 | (1,32,0,0,0,0,1) | 32 | 0 | 0 | 0 | 97         | 126,7  | 141,7  | No           | No           |
|        |          |        | 4.4 | (1,64,0,0,0,0,1) | 64 | 0 | 0 | 0 | 193        | 126,1  | 141,1  | No           | No           |
| 8      | 1        | 0001   | 8.1 | (1,8,0,0,0,0,1)  | 8  | 0 | 0 | 0 | 25         | 135,3  | 152,4  | No           | No           |
|        |          |        | 8.2 | (1,16,0,0,0,0,1) | 16 | 0 | 0 | 0 | 49         | 124,4  | 139,3  | No           | No           |
|        |          |        | 8.3 | (1,32,0,0,0,0,1) | 32 | 0 | 0 | 0 | 97         | 124,3  | 138,9  | No           | No           |
|        |          |        | 8.4 | (1,64,0,0,0,0,1) | 64 | 0 | 0 | 0 | 193        | 124,3  | 138,8  | No           | No           |

Notas: El comportamiento visto durante el entrenamiento de los modelos confirma lo dicho por (Ramos,Oliveira & Silva, 2014) que una red neuronal tipo secuencial de orden (n,u,v,1) es equivalente a un modelo AR(1).  
Esto debido que los modelos AR(1)+RNA se aprenden el comportamiento de la predicción hecha por el modelo anterior durante las fases de entrenamiento y validación.

Fuente: Elaboración propia con Datos del Código en Python.

#### 3.1.4. Iteraciones sobre Modelos

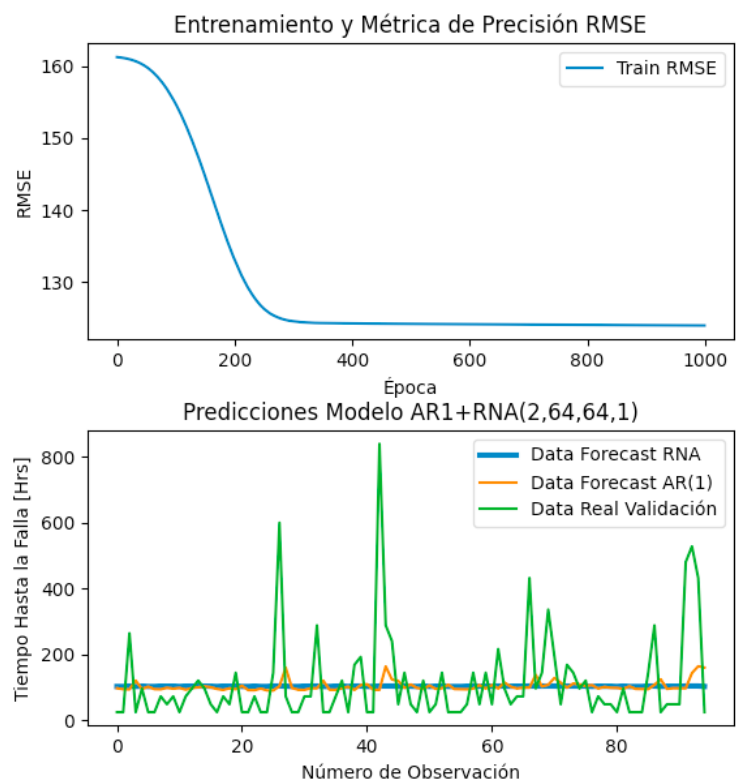
En este punto del presente trabajo de titulación surge la siguiente hipótesis: Incluir variables de entrada de modelación de confiabilidad a un modelo  $AR(1)+RNA(E,O,S)$  mejora la métrica de precisión y logra predecir eventos de fallas.

Es por esto que a los modelos seleccionados se le agregan entradas como: función confiabilidad, densidad de fallas y tasa de fallas; cada una de ellas en función del tiempo hasta la falla de los eventos observados en el equipo.

#### 3.1.5. Resultados de la Iteración

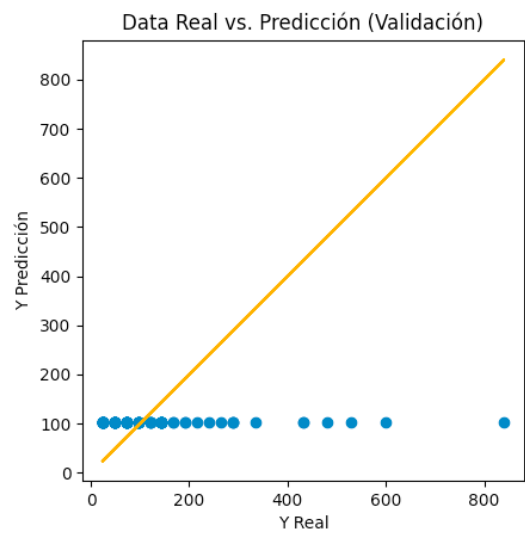
A continuación, se presentan algunos de los resultados obtenidos del total de 624 iteraciones realizadas:

- El modelo n° 10.5:  $AR1+RNA(2,64,64,1)$  de 185 parámetros, se valida con un RMSE de 138,5 significa que no supera al modelo  $AR(1)$  y no logra realizar predicciones, solo establece una línea base cercano a un tiempo de 100 hrs hasta la falla tal como en la Figura 3-1. La visualización de los valores que predice el modelo se encuentran en la Figura 3-2.
- El modelo n° 11.19:  $AR1+RNA(3,32,8,16,1)$  de 553 parámetros, se valida con un RMSE de 136,6 superando al modelo  $AR(1)$  y Weibull, pero no logra una predicción.
- El modelo n° 7.28:  $AR1+RNA(3,64,32,16,1)$  de 2.881 parámetros, se valida con un RMSE de 95,7 logrando una predicción temprana de los eventos de fallas.
- El modelo 15.256:  $AR1+RNA(4,64,64,64,1)$  de 12.865 parámetros se valida con un RMSE de 41,4 logrando una predicción temprana de los eventos de fallas.
- El modelo n° 7.64:  $AR1+RNA(3,64,64,64,1)$  de 8.641 parámetros se valida con un RMSE de 40,4 logrando una predicción temprana de los eventos de fallas.
- El modelo 12.16:  $AR1+RNA(2,64,64,1)$  de 4.417 parámetros se valida con un RMSE de 33,9 siendo el menor valor obtenido. Supera al modelo  $AR(1)$  y Weibull Logrando una predicción temprana de los eventos de fallas.



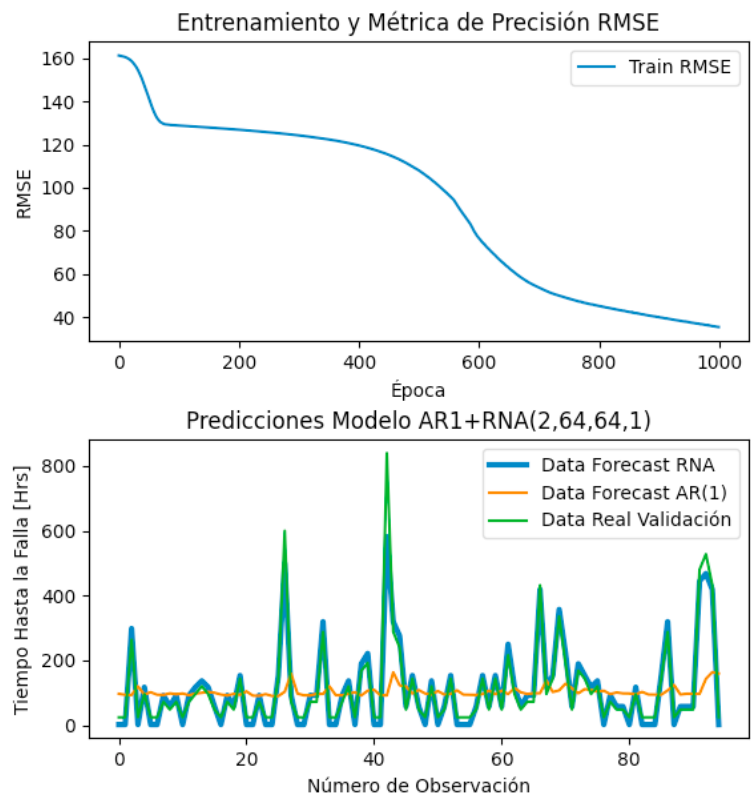
Fuente: Elaboración Propia con Python.

Figura 3-1: Validación Modelo 10.5 AR1+RNA(2,64,64,1).



Fuente: Elaboración Propia con Python.

Figura 3-2: Data Real vs. Predicción en Modelo 10.5.

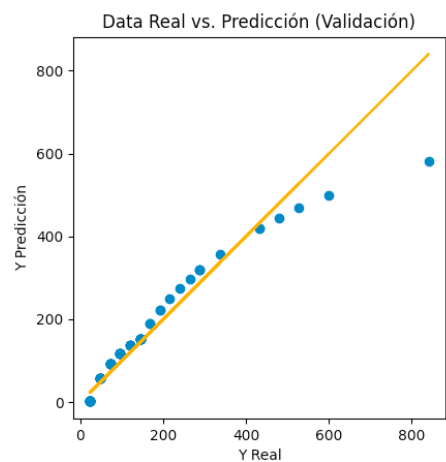


Fuente: Elaboración Propia en Python.

Figura 3-3: Validación Modelo 12.6 AR1+RNA(2,64,64,1).

Respecto a la calidad en predicción mostrada en la Figura 3-4, se puede indicar que el modelo 12.6:

- Es excelente para eventos de corto plazo.
- Es casi exacto para eventos de mediano plazo.
- Predice con 200 horas de anticipación eventos de largo plazo.



Fuente: Elaboración Propia con Python.

Figura 3-4: Data Real vs. Predicción del Modelo 12.6.

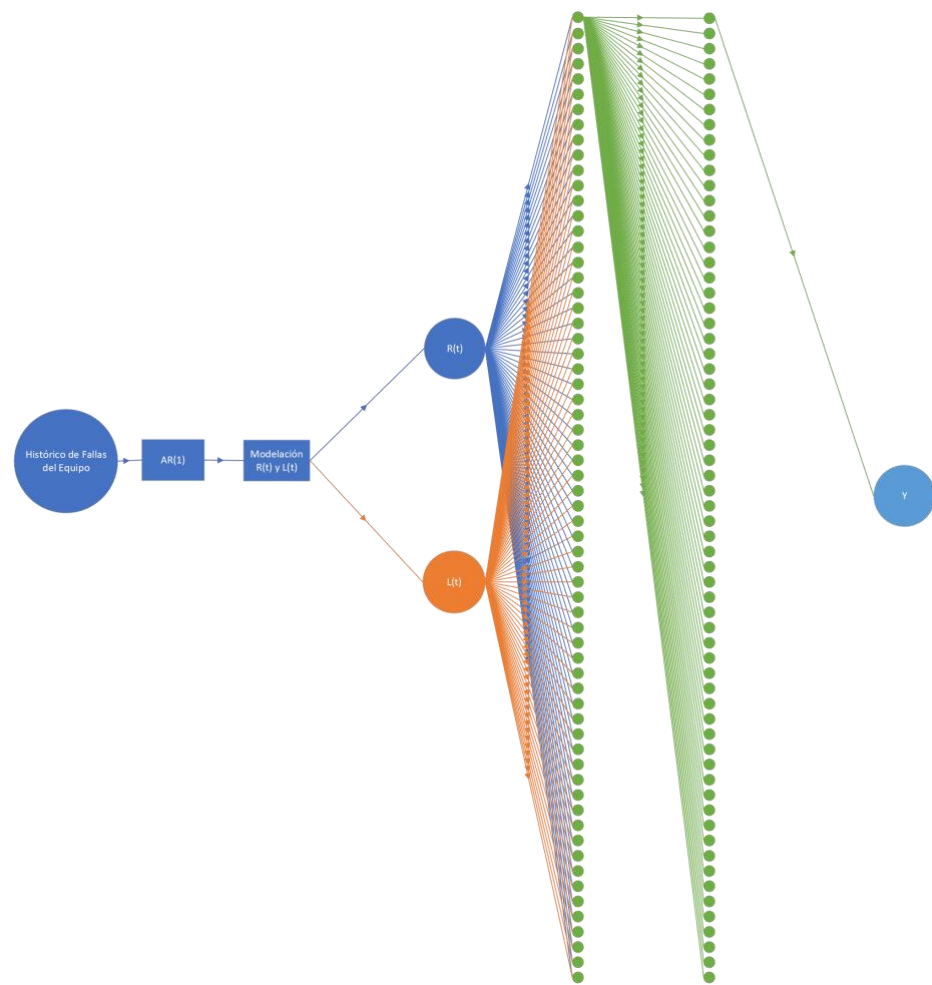
3.2.     **MODELO FINAL PROPUESTO**

Tras comparar los 624 modelos elaborados, se propone que la empresa Celulosa Arauco utilice el modelo 12.16:  $AR1+RNA(2,64,64,1)$  de 4.417 parámetros que fue validado con un RMSE de 33,9 siendo el menor valor obtenido.

El modelo supera al modelo  $AR(1)$  reduciendo la métrica de precisión RMSE en un 75,3%. También supera al modelo Weibull en un 94,9% y en un 75,1 al modelo base propuesto.

Lo interesante es que las entradas de la Red Neuronal Artificial resultante son: Confiabilidad o  $R(t)$ , junto con Tasa de Fallas o  $\lambda(t)$ . Logrando además una simplificación del modelo de acuerdo al contexto operacional del equipo y los datos analizados del histórico de fallas.

En la Figura 3-5 se muestra de manera parcial cómo luce el modelo final propuesto en el presente trabajo. Primero se debe utilizar la parte  $AR(1)$  para encontrar los pronósticos de eventos de fallas para obtener las entradas de la red neuronal artificial mediante la modelación de la confiabilidad y tasa de fallas respecto al tiempo.



Fuente: Elaboración Propia con Visio.

Figura 3-5: Ejemplo Red Neuronal Propuesta.

3.3. ANÁLISIS DE COSTOS E IMPACTO DEL MODELO

Si el costo de ineficiencia es de 7.694,4 [USD/Hr] respecto al costo de estar fuera de servicio el equipo. Significa que para el año 2016 la empresa tuvo un costo de 136.706,4 USD. Mientras que en el año 2017 fue de 77.459,5 USD. Y para el año 2018 de 51.806,4 USD. Lo anterior se obtiene desde la base de datos facilitada por la empresa para su análisis y corresponden al equipo con tag 1001: Descortezador de tambor.

El modelo propuesto impactaría directamente en esos costos al lograr una predicción de los eventos de fallas en corto, mediano y largo plazo. Permitiendo que la gerencia de mantenimiento tener una idea de cuando ocurrirán las detenciones en el equipo. Esto facilita la ejecución de actividades de mantenimiento para evitar el costo de ineficiencia lo que genera un ahorro considerable para la empresa.

## **CONCLUSIONES**

Existen diversas técnicas que pueden ser utilizadas perfectamente en la industria para diferentes tipos de problemáticas. Las Redes Neuronales Artificiales son uno de los algoritmos de machine learning más populares, seguido de Random Forests. Lamentablemente en (Baptista, Sankararaman, de Medeiros, Nascimento Jr., & Henriquesa, 2018) fue una de las técnicas con peor desempeño en los resultados del modelo general propuesto. Es por esto que fue seleccionado como candidato las Redes Neuronales Artificiales, logrado la definición de los conceptos asociados al modelo ARMA y RNA.

Se realizan 624 iteraciones para determinar cual es el modelo óptimo para lograr una predicción de eventos de fallas antes de que estos ocurran y con el mínimo valor de la métrica de precisión RMSE. Se itera sobre el número de entradas, cantidad de capas y neuronas por capa.

Se determina que el modelo que minimiza el RMSE es un  $AR(1)+RNA(2,64,64,1)$  con un RMSE de validación de 33,9. Logrando una mejora del 75,3% respecto al modelo  $AR(1)$ . También supera al modelo Weibull en un 94,9% y en un 75,1 al modelo base propuesto.

Para futuros trabajos, se podría comparar los resultados: 1) Contra un modelo puro de Redes Neuronales Recurrentes, RNN o sólo cambiando el tipo de red. 2) Contra el modelo de (Mulyana, 2019) que utiliza una Red Neuronal Auto Regresiva con entrada Exógena, NNARX o sólo intercambiando el tipo de red. 3) Inclusive sería interesante comparar el modelo propuesto como en (As'ad, Sujito, & Setyowibowo, 2020) que utiliza una Red Neuronal Auto Regresiva, NNAR(p,k) con p-rezagos y k-capas ocultas.

## **BIBLIOGRAFÍA**

- Chollet, F. (2018). *Deep Learning with Python* (Primera ed.). Nueva York: Manning Publications Co.
- Carvalho, T. P., Soares, F. A., Vita, R., Francisco, R. d., Basto, J. P., & Alcala, S. G. (Noviembre de 2019). A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137(106024).
- Baptista, M., Sankararaman, S., de Medeiros, I. P., Nascimento Jr., C., & Henriquesa, E. M. (Enero de 2018). Forecasting Fault Events for Predictive Maintenance using Data-driven Techniques and ARMA modeling. *Computers and Industrial Engineering*, 115, 41-53.
- Ramos, P., Oliveira, J. M., & Silva, P. (2014). Predictive maintenance of production equipment based on neural network autoregression and ARIMA. *Universidade do Porto*, 10.
- Brownlee, J. (Agosto de 2021). *Machine Learning Mastery Scientific Community Online*. Obtenido de How to Grid Search ARIMA Model Hyperparameters with Python: <https://machinelearningmastery.com/grid-search-arima-hyperparameters-with-python/>
- Ponce Cruz, P. (2010). *Inteligencia Artificial con Aplicaciones a la Ingeniería* (Primera ed.). México: Alfaomega Grupo Editor.
- Mora, A. (2009). *Mantenimiento - Planeación, ejecución y control*. México DF: Alfaomega Grupo Editor, S.A.
- Mulyana, T. (2019). Identification of heat exchanger by neural network autoregressive with exogenous input model. *Intechopen*, 23.
- As'ad, M., Sujito, & Setyowibowo, S. (2020). Prediction of daily gold prices using an autoregressive neural network. *Jurnal Ilmiah Bidang Teknologi Informatika dan Komunikasi*, 5.

## ANEXO A

### Métricas de Desempeño (Precisión)

- **MAE: Mean Absolute Error:** Expresa la media de los valores absolutos del error.

$$MAE = \frac{1}{n} \sum |\hat{y} - y|$$

- **RMSE: Root Mean Squared Error:** Expresa la media del error al cuadrado.

$$RMSE = \sqrt{\frac{1}{n} \sum (\hat{y} - y)^2}$$

- **MAPE: Mean Absolute Percentage Error:** Expresa la media del porcentaje de error en relación con el valor real.

$$MAPE = \frac{1}{n} \sum \left| \frac{\hat{y} - y}{y} \right|$$

- **SSD: Sample Standard Deviation:** Expresa la desviación estándar muestral.

$$S = \sqrt{\frac{\sum (\hat{y} - \bar{y})^2}{n - 1}}$$

**ANEXO B**

Historia de las RNA

| Año            | Acontecimiento                                                                                                                                                                                                                        |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 384-322 a.C.   | Se podría considerar que los primeros pasos de la IA fueron dados por Aristóteles, cuando se dispuso a explicar y codificar ciertos estilos de razonamiento deductivo llamado silogismos.                                             |
| d.C. 1235-1316 | Ramon Llull, místico y poeta catalán construyó un conjunto de ruedas llamado Ars Magna, el cual iba a ser una maquina capaz de responder todas las preguntas.                                                                         |
| 1646-1716      | Gottfried Leibniz, quería un algebra universal en el cual todos los conocimientos puedan algún día ser interpuestos dentro de un sistema deductivo único.                                                                             |
| 1854           | George Boole, comenzó a desarrollar los fundamentos de la lógica proposicional.                                                                                                                                                       |
| 1879           | Gottlob Frege propuso un sistema de notación para el razonamiento mecánico conocido como cálculo proposicional (lógica matemática moderna).                                                                                           |
| 1943           | Introducidas por McCulloch y Walter Pitts, escribieron teorías acerca de las relaciones entre los elementos del cálculo simple y neuronas biológicas. Sólo servían como aproximaciones a funciones booleanas (AND, OR y NOT).         |
| 1960           | Frank Rosenblatt, exploro el uso de redes llamadas perceptrones. Se introdujo la función “sigmoide”, que tiene un valor umbral de 0,5. Tenía que ser una función matemática fácil de calcular.                                        |
| 1965           | Ivankhenko, Perceptrón Multicapa.                                                                                                                                                                                                     |
| 1969           | Minsky, Problema XOR, ya que el perceptrón no puede simular la puerta lógica XOR, demostrando los límites del perceptrón. Las posibles soluciones son: en vez de utilizar rectas, utilizar curvas o también utilizar varias neuronas. |
| 1975           | Werbos, Algoritmo de Backpropagation, primeros entrenamientos de redes neuronales automáticamente, todavía sin capacidad de computo ideal.                                                                                            |
| 1980           | Se empezó a retomar la estructura neuronal, convirtiéndose en uno de los mas populares y exitosos subcampos de la Inteligencia Artificial.                                                                                            |

Fuente: Elaboración propia en base a bibliografía consultada.

## ANEXO C

### Ejemplo de Regla de la Cadena en Redes Neuronales

#### Derivadas de la Función de Activación:

TanH:

$$g'(z) = 1 - \tanh^2(z)$$

ReLU:

$$\begin{cases} 0: z < 0 \\ ? : z = 0 \\ 1: z > 0 \end{cases}$$

La derivada en 0 queda indeterminada, normalmente se trabaja suponiendo que la probabilidad de que la función devuelva un valor 0 es demasiado baja.

LeakyReLU:

$$\begin{cases} n: z < 0 \\ ? : z = 0 \\ 1: z > 0 \end{cases}$$

La derivada del sigmoide:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial \hat{y}}{\partial z} = \frac{1}{1 + e^{-z}} * \left(1 - \frac{1}{1 + e^{-z}}\right) = \hat{y}(1 - \hat{y})$$

Entonces:

$$\begin{aligned} \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} &= \left(\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}\right) * \hat{y}(1 - \hat{y}) = -\frac{y\hat{y}(1 - \hat{y})}{\hat{y}} + \frac{(1 - y)\hat{y}(1 - \hat{y})}{(1 - \hat{y})} \\ &= y(1 - \hat{y}) + (1 - y)\hat{y} = -y + y\hat{y} - y\hat{y} = \hat{y} - y \end{aligned}$$

En la primera capa:

$$z = w_1 * x_1 + \dots + w_n * x_n + b$$

$$\frac{\partial z}{\partial w_1} = x_1 \quad ; \quad \frac{\partial z}{\partial w_n} = x_n \quad ; \quad \frac{\partial z}{\partial b} = 1$$

Entonces:

$$\begin{aligned} \frac{\partial L(\hat{y}, y)}{\partial w_n} &= \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial w_n} = (\hat{y} - y) * w_n \\ \frac{\partial L(\hat{y}, y)}{\partial b} &= \frac{\partial L(\hat{y}, y)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial b} = (\hat{y} - y) \end{aligned}$$

**ANEXO D****Código “EjecutaWeibull”**

```

import pandas as pd
import numpy as np
import statistics
import matplotlib.pyplot as plt
import scipy.special
from sklearn.metrics import mean_squared_error

def fweibull(a, b, t):
    f = (b / a) * ((t / a) ** (b - 1)) * np.exp(-(t / a) ** b)
    return f

def Fweibull(a, b, t):
    F = 1 - np.exp(-(t / a) ** b)
    return F

def TFweibull(a, b, t):
    TF = (fweibull(a, b, t)) / (1 - Fweibull(a, b, t))
    return TF

# Importar DATA:
df = pd.read_excel(r'DATA0.xlsx').sort_values(by='Delta [Hrs]')

# Estimación de Parámetros Dist.Weibull:
n = len(df)                                # Numero muestral
df['kj'] = range(1,n+1)                    # SIN ranking ajustado
df['Ft_b'] = (df['kj']+0.3)/(n+0.4)         # F(t): Rango Mediana de Benard
df['Rt_b'] = 1-df['Ft_b']                  # R(t): Confiabilidad

# Listas con la Info Necesaria:
kj = df['kj'].values.tolist()
Ti_l = df['Delta [Hrs]'].values.tolist()
Rt_b = df['Rt_b'].values.tolist()

# Ejes del ploteo
Ejex = [np.log(Ti_l[i]) for i in range(len(kj))]
Ejey = [np.log(-(np.log(Rt_b[i]))) for i in range(len(kj))]

# Pendiente e Intersección: Método Mínimos Cuadrados
nr = len(Ejex)
sumx, sumy = sum(Ejex), sum(Ejey)
XY = [Ejex[i]*Ejey[i] for i in range(nr)]
X2 = [Ejex[i]**2 for i in range(nr)]
Y2 = [Ejey[i]**2 for i in range(nr)]
sumxy, sumx2 = sum(XY), sum(X2)

```

```

# Coef. de Correlación y Coef. de Determinación R^2
# Cuadrado del Coeficiente de correlación de Pearson (rxy)^2:
destx = statistics.stdev(Ejex)
desty = statistics.stdev(Ejey)
rxy = (sumxy-nr*(sumx/nr)*(sumy/nr))/((nr-1)*destx*desty)
R2 = rxy**2

# Intersección (A) y Pendiente (B):
B = (nr * sumxy - sumx * sumy) / (nr * sumx2 - (sumx ) ** 2)
A = (sumy - B * sumx ) / nr
print('RESULTADOS DEL AJUSTE WEIBULL')
print('La ecuación de la Recta es: ',B,'X +',A)
print('Coeficiente de Determinación R^2: ',R2)
Ec = [A+B*Ejex[i] for i in range(len(Ejex))]

# Estimación de Parámetros Weibull:
Alpha = np.exp(-A / B)
Beta = B
MTBF= Alpha*scipy.special.gamma(1+(1/Beta))

print('El valor de Alpha es: ',Alpha)
print('El valor de Beta es: ', Beta)
print('El valor de MTBF es: ', MTBF)

# Ploteo del Grafico Dispersion Weibull:
plt.scatter(Ejex,Ejey,alpha=0.3,label='Datos',c='orange')
plt.plot(Ejex,Ec,label='Ajuste Lineal')
plt.title('Gráfico de Dispersión Weibull')
plt.xlabel('Ln(ti)')
plt.ylabel('Ln(-Ln(R(ti)))')
plt.legend()
plt.show()
plt.clf()

# Creación de números aleatorios:
datos = np.random.rand(98)
alpha = Alpha
beta = Beta
y_predicted = (list(alpha * (-np.log(1 - datos)) ** (1 / beta)))
df2 = pd.read_excel(r'DATA0.xlsx')
y_actual = df2['Delta [Hrs]'].values.tolist()
rms = mean_squared_error(y_actual, y_predicted, squared=False)
print('El RMSE es de: ', rms)
y_predicted.sort()

# Análisis Exploratorio de Datos:
plt.title('Histograma de datos simulados')
plt.hist(y_predicted, bins=10, color='green', alpha=0.5)
plt.ylabel('Frecuencia Absoluta')
plt.xlabel('Tiempo (ti)')

```

```
# plt.xlim(0,500)
# plt.ylim(0,250)
plt.show()
plt.clf()

Equis = np.arange(1, 300)
plt.title('Tasa de Fallas y Confiabilidad de los datos simulados')
plt.plot(Equis, TFweibull(alpha, beta, Equis), label='Tasa de Fallas')
plt.ylabel('Tasa de Fallas')
plt.twinx()
plt.plot(Equis, (1 - Fweibull(alpha, beta, Equis)) * 100, 'orange',
label='Confiabilidad')
plt.ylabel('Confiabilidad')
plt.legend(('C'), loc='best')
plt.show()
plt.clf()
```

**ANEXO E****Código “EjecutaBROWNLEEbased”**

```

import pandas as pd
import matplotlib.pyplot as plt
import warnings
from math import sqrt
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error

# Importar Data
ndf=pd.read_excel(r'DATA0.xlsx')
X=ndf['Evento']
Y=ndf['Delta [Hrs]']

# Plot Dataframe
plt.plot(X,Y)
plt.ylabel('Tiempo Hasta la Falla [Hrs]')
plt.xlabel('Número de Evento')
plt.title('Tiempo Hasta la Falla del Equipo 1002-L1')
plt.show()
plt.clf()

# Parametros
p_values = [1, 2, 3]
d_value = [0]
q_values = range(0, 3)
warnings.filterwarnings("ignore")

# Evaluar un modelo ARIMA dado un orden (p,d,q)
predictions = []
warnings.filterwarnings("ignore")
best_score, best_cfg = float("inf"), None
for p in p_values:
    for d in d_value:
        for q in q_values:
            order=(p,d,q)
            try:
                # Lista train y test.
                Y = [float(x) for x in Y]
                train=[]
                test=[]
                for i in range(len(Y)):
                    if X[i]%2==0:
                        test.append(Y[i])
                    elif X[i]%2!=0:
                        train.append(Y[i])
                history = [x for x in train]

```

```

        print('PROCESANDO ARIMA: ', order)
        for t in range(len(test)):
            model = ARIMA(history, order=order)
            model_fit = model.fit(dispatch=0)
            yhat = model_fit.forecast()[0]
            predictions.append(yhat)
            obs = test[t]
            history.append(obs)

        # evaluate forecasts
        rmse = sqrt(mean_squared_error(test, predictions))
        if rmse < best_score:
            best_score, best_cfg = rmse, order
            print('ARIMA%s RMSE=%.3f' % (order, rmse)) # imprime arima string
            y rmse float.
        except:
            continue
    print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))

```

**ANEXO F****Código “EjecutaARIMA1”**

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt

def fweibull(a, b, t):
    f = (b / a) * ((t / a) ** (b - 1)) * np.exp(-(t / a) ** b)
    return f

def Rweibull(a, b, t):
    F = np.exp(-(t / a) ** b)
    return F

def TFweibull(a, b, t):
    TF = (fweibull(a, b, t)) / Rweibull(a, b, t)
    return TF

# Import Data
df=pd.read_excel(r'DATA0.xlsx')
X=df['Evento']
Y=df['Delta [Hrs]']
warnings.filterwarnings("ignore")

# Estimación de Parámetros Dist.Weibull:
n = len(df)                                # Numero muestral
df['kj'] = range(1,n+1)                    # SIN ranking ajustado
df['Ft_b'] = (df['kj']+0.3)/(n+0.4)         # F(t): Rango Mediana de Benard
df['Rt_b'] = 1-df['Ft_b']                  # R(t): Confiabilidad

# Listas con la Info Necesaria:
kj = df['kj'].values.tolist()
Ti_l = df['Delta [Hrs]'].values.tolist()
Rt_b = df['Rt_b'].values.tolist()

# Ejes del ploteo
Ejex = [np.log(Ti_l[i]) for i in range(len(kj))]
Ejey = [np.log(-(np.log(Rt_b[i]))) for i in range(len(kj))]

# Pendiente e Intersección: Método Mínimos Cuadrados
nr = len(Ejex)
sumx,sumy = sum(Ejex), sum(Ejey)
XY = [Ejex[i]*Ejey[i] for i in range(nr)]

```

```

X2 = [Ejex[i]**2 for i in range(nr)]
Y2 = [Ejey[i]**2 for i in range(nr)]
sumxy, sumx2 = sum(XY), sum(X2)

# Intersección (A) y Pendiente (B):
B = (nr * sumxy - sumx * sumy) / (nr * sumx2 - (sumx ) ** 2)
A = (sumy - B * sumx ) / nr
# Estimación de Parámetros Weibull:
Alpha = np.exp(-A / B)
Beta = B

# Add cols
df['f(t)']=fweibull(Alpha,Beta,df['Delta [Hrs]'])
df['R(t)']=Rweibull(Alpha,Beta,df['Delta [Hrs]'])
df['TF']=TFweibull(Alpha,Beta,df['Delta [Hrs]'])
# cols to list
ft=df['f(t)'].values.tolist()
rt=df['R(t)'].values.tolist()
tfallas=df['TF'].values.tolist()

#Extracción datos:
train=[]      #Lista Entrenamiento
test=[]       #Lista Validación
test2=[float(x) for x in Y]
for i in range(len(Y)):
    if X[i]%2==0:
        test.append(Y[i])
    elif X[i]%2!=0:
        train.append(Y[i])
history = [x for x in test]

predictions = []
# walk-forward validation
for t in range(len(test)):
    model = ARIMA(history, order=(1,0,0))
    model_fit = model.fit(dispatch=0)
    yhat = model_fit.forecast(1)[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
# evaluate forecasts
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)

# plot forecasts against actual outcomes
plt.plot(test)
plt.plot(predictions, color='red')
plt.title("Validación de Modelo ARIMA(1,0,0) o AR(1)")
plt.xlabel("Número de Observación")

```

```

plt.ylabel("Tiempo Hasta la Falla")
plt.show()
plt.clf()

predictions2=[]
for t in range(len(test2)):
    model = ARIMA(history, order=(1,0,0))
    model_fit = model.fit(dispatch=0)
    yhat = model_fit.forecast(1)[0]
    predictions2.append(yhat)
    obs = test2[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
# evaluate forecasts
rmse2 = sqrt(mean_squared_error(test2, predictions2))
print('Test RMSE: %.3f' % rmse2)

forecast=model_fit.forecast(steps=2)[0][1] # Predice el último y siguiente evento
de falla
print('Forecast: ',forecast)

# plot forecasts against actual outcomes
plt.plot(test2)
plt.plot(predictions2, color='red')
plt.title("Modelo ARIMA(1,0,0) o AR(1) Serie Completa")
plt.xlabel("Número de Observación")
plt.ylabel("Tiempo Hasta la Falla")
plt.show()
plt.clf()

# Listas a dataframe to csv
Ev=[x for x in X]
Delta=[y for y in Y]
Arima=[z[0] for z in predictions2]

next=len(Ev)+1
Ev.append(next)
Delta.append(forecast)

dataframe=pd.DataFrame(list(zip(Ev,Delta,Arima,ft,rt,tfallas)),
                        columns=['Evento','Delta [Hrs]','Forecast
AR(1)','f(t)','R(t)','TF'])
print(dataframe)

dataframe.to_csv('DataRNA.csv', index=False, encoding='utf-8')

```

**ANEXO G****Código “EjecutaRNA”**

```

import keras
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.layers.core import Dense

# Funciones:
def Red_1capa(input,neuronas,act):
    model.add(Dense(neuronas, input_shape=[input], activation=act, use_bias=True))
    model.add(Dense(1))
    return model

def Red_2capas(input,neuronas1,neuronas2,act):
    model.add(Dense(neuronas1, input_shape=[input], activation=act, use_bias=True))
    model.add(Dense(neuronas2, activation=act, use_bias=True))
    model.add(Dense(1))
    return model

def Red_3capas(input,neuronas1,neuronas2,neuronas3,act):
    model.add(Dense(neuronas1, input_shape=[input], activation=act, use_bias=True))
    model.add(Dense(neuronas2, activation=act, use_bias=True))
    model.add(Dense(neuronas3, activation=act, use_bias=True))
    model.add(Dense(1))
    return model

def Red_4capas(input,neuronas1,neuronas2,neuronas3,neuronas4,act):
    model.add(Dense(neuronas1, input_shape=[input], activation=act, use_bias=True))
    model.add(Dense(neuronas2, activation=act, use_bias=True))
    model.add(Dense(neuronas3, activation=act, use_bias=True))
    model.add(Dense(neuronas4, activation=act, use_bias=True))
    model.add(Dense(1))
    return model

def Red_5capas(input,neuronas1,neuronas2,neuronas3,neuronas4,neuronas5,act):
    model.add(Dense(neuronas1, input_shape=[input], activation=act, use_bias=True))
    model.add(Dense(neuronas2, activation=act, use_bias=True))
    model.add(Dense(neuronas3, activation=act, use_bias=True))
    model.add(Dense(neuronas4, activation=act, use_bias=True))
    model.add(Dense(neuronas5, activation=act, use_bias=True))
    model.add(Dense(1))
    return model

# Importar y Utilizar como lista la salida de AR(1):
df=pd.read_csv('DataRNA.csv') # Importa el archivo
df['Forecast**2']=df['Forecast AR(1)']**2 # Aumento de dimensión Forecast AR(1)
#print(df.columns.values)
#print(df.head(5))
#print(df.shape)

```

```

# Tomar muestra de 5 elementos del Conjunto de datos:
dataset = df.values
datos = dataset[:, :]
datos_v=dataset[:, :]

# Conjuntos de Entrenamiento y Validación:
AR_data=dataset[:-5, 2]
X_train=dataset[:-5, [3, 5]]          ##### SEGUNDO Cambiar entradas
Y_train=dataset[:-5, 1]

AR_datatest=datos_v[:, 2]
X_test=datos_v[:, [3, 5]]             ### TERCERO Coincidir con X_train
Y_test=datos_v[:, 1]

# Variables de Iteración:
epocas=1000
rango_epocas=range(epocas)
# Entrenamiento del Modelo:
model = keras.Sequential()
funcion_activacion = tf.keras.layers.LeakyReLU(alpha=0.001)
Red_2capas(2, 8, 16, funcion_activacion)          ##### PRIMERO
Cambiar entradas, neuronas y red
opt = tf.keras.optimizers.Adam(lr=1e-3, decay=1e-5)
model.compile(optimizer=opt, loss='MeanSquaredError',
metrics=[tf.keras.metrics.RootMeanSquaredError()])
historial=model.fit(X_train, Y_train, epochs=epocas, verbose=1)

# Evalua el modelo sobre conjunto de validación:
print('EVALUACIÓN DEL MODELO:')
model.evaluate(X_test, Y_test, verbose=1)
model.summary()

# Guarda el modelo:
model.save('Modelo 10-5_n')                ### CUARTO Cambiar
Nombre del modelo

# Calcula RMSE como la raíz de MSE
fperdida=[np.sqrt(i) for i in historial.history['loss']]

predicciones=model.predict(X_test)
#print(predicciones)
forecast_RNA=[predicciones[i][0] for i in range(len(predicciones))]
equis=range(len(X_test))

plt.subplot(2, 1, 1)
plt.plot(rango_epocas, fperdida, label='Train RMSE')
plt.title('Entrenamiento y Métrica de Precisión RMSE')
plt.ylabel('RMSE')
plt.xlabel('Época')

```

```
plt.legend()
plt.subplot(2,1,2)
plt.plot(equis,forecast_RNA, label='Data Forecast RNA',linewidth=3)
plt.plot(equis,AR_datatest, label='Data Forecast AR(1)')
plt.plot(equis,Y_test, label='Data Real Validación')
plt.title('Predicciones Modelo AR1+RNA(2,64,64,1)')
plt.ylabel('Tiempo Hasta la Falla [Hrs]')
plt.xlabel('Número de Observación')
plt.legend()
plt.show()
plt.clf()

flatten_pred=predicciones.flatten()
plt.axes(aspect='equal')
plt.scatter(Y_test,flatten_pred)
plt.title('Data Real vs. Predicción (Validación)')
plt.xlabel('Y Real')
plt.ylabel('Y Predicción')
plt.plot(Y_test,Y_test,color='orange')
plt.show()
plt.clf()
```