Application of Attention Mechanism in Deep Neural Network Architecture for System Failure Prognostics

Hamid Reza Behizadi

A Thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Master of Computer Science at
Concordia University
Montréal, Québec, Canada

May 2023

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Hamid Reza Behizadi**

Entitled: **Application of Attention Mechanism in Deep Neural Network Architecture for System Failure Prognostics**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Juergen Rilling

_____ Examiner
Dr. Juergen Rilling

_____ Examiner
Dr. Yann-Gael Gueheneuc

_____ Thesis Co-supervisor
Dr. Leila Kosseim

_____ Thesis Co-supervisor
Dr. Jia Yuan Yu

Approved by _____
Dr. Leila Kosseim, Graduate Program Director

May 5, 2023 _____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Application of Attention Mechanism in Deep Neural Network Architecture for System Failure Prognostics

Hamid Reza Behizadi

Machine health monitoring and management are essential improvements that must be considered in the industry toward smart manufacturing. Intelligent prognosis and health management (PHM) systems have demonstrated remarkable capabilities for industrial use and, consequently, have become active research areas in the last several decades. Predictive Maintenance (PM) generally predicts faults or breakdowns in a deteriorating system to optimize maintenance efforts by evaluating the system's status using historical data. In this strategy, the Remaining Useful Life (RUL) of the components is anticipated using characteristics, which typically include sensors and operational profiles.

This research aims to evaluate the possibility of predicting the RUL of a system based on sensor data by deploying an attention-based deep learning model. RUL prediction based on the attention mechanism is a relatively new approach with promising results. One advantage of this approach is that it can be useful for interpreting the results and understanding the underlying factors contributing to the RUL. Applying an attention mechanism to find temporal dependencies also shows improvement in model performance by detecting the most important part of the sequences to be passed to the prediction model.

Our proposed model has shown a noticeable impact on the performance of the neural network architecture from the attention mechanism added to the pipeline by keeping the model light in terms of computational resources and training time. The proposed model clearly shows the attention mechanism's high impact on predicting sequential data. This technique can also be used in more complex ensemble-based architectures to improve performance.

Acknowledgments

I would like to express my deepest appreciation to Dr. Leila Kosseim, for all the guidance, support, and dedication throughout my thesis completion. Also, I appreciate all help from my co-supervisor, Dr. Jia Yuan Yu. Their knowledge and insightful instructions carried me through my study.

I am also grateful to my dear mother, father, and sister, for their love and encouragement during my life. Their constant support has given me courage and inspiration throughout my academic journey. I could not have undertaken this journey without the help of my kind wife, Mahdis. Her love, patience, and understanding have been invaluable in helping me overcome the challenges of writing this thesis.

This thesis would not have been achievable without the help and inspiration of these wonderful people. Please accept my sincere gratitude.

Contents

List of Figures

List of Tables

Abbreviations

ALSTM Attention-based Long Short-term Memory

ANN Artificial Neural Network

Bi-LSTM Bidirectional Long Short-term Memory

C-MAPSS Commercial Modular Aero-Propulsion System Simulation

CBM Condition-based Maintenance

CNN Convolutional Neural Network

DBN Deep Belief Network

DBN-FNN Deep Belief Network Feedforward Neural Network

EIS Engineering Immune System

FC Fully Connected

FNN Feedforward Neural Network

GA Genetic Algorithm

GRU Gated Recurrent Unit

HI Health Index

HPC High-Pressure Compressor

HPT High-Pressure Turbine

IBM Interval-based Maintenance

ICT Information and Communication Technologies

IIOT Industrial Internet of Things

ISHM Integrated System Health Monitoring

LPC Low-Pressure Compressor

LPT Low-Pressure Turbine

LSTM Long Short-Term Memory

MLP Multilayer Perceptron Network

MODBNE Multi-objective Deep Belief Network Ensemble

MSCNN Multiscale Convolutional Neural Network

MSE Mean Squared Error

PDF Probability Density Function

PHM Prognostic Health Management

PM Preventive Maintenance

RBM Restricted Boltzmann Machine

RCM Reliability-centered Maintenance

ReLU Rectified Linear Unit

RF Random Forest

RM Reactive Maintenance

RMS-prop Root Mean Squared Propagation

RMSE Root-mean-square Error

RNN Recurrent Neural Network

RUL Remaining Useful Life

SGD Stochastic Gradient Descent

SVR Support Vector Regression

TFR Time-frequency Representation

TSMC-CNN Time Series Multiple Channel Convolutional Neural Network

TW Time Window

Chapter 1

Introduction

Over time, human beings have had to cope with various damages and failures in their projects related to material degradation, structural weariness, or even poor design decisions. Hence, engineering diagnostics and maintenance are essential in different business sectors, including heavy industries, aerospace, manufacturing, and automotive. With the development of technology over the past few decades, maintenance duties have evolved dramatically. Having more developed and powerful systems plus keeping track of historical data allows further advancement in diagnostics by predicting the degradation far ahead of their occurrence. This is where prognostics come into play and can lead the industry to the next generation of maintenance. This generation change will likely let predictive maintenance contribute much more and gradually replace older maintenance approaches.

1.1 Background and Motivation

1.1.1 Smart Manufacturing

Smart manufacturing is a rising global need and strategic urgency for developing advanced, innovative, and reliable predictive health management systems. Furthermore, with the

growth of the Industrial Internet of Things (IIoT), smart maintenance has become a more important tool. Consequently, gathering data and integrating devices' roles have become crucial to make systems more knowledgeable about smart manufacturing. Maintenance is a collection of technical, managerial, and administrative actions designed to guarantee a system's optimal performance [1]. In particular, Prognostic Health Management (PHM) is a maintenance strategy focused on forecasting the possibility of component failure and, as a result, limiting unplanned downtimes in complex systems. Enhancing maintenance effectiveness, safe functionality, and performance during this procedure is critical [2]. An overview of the different components and their connections in a predictive health management system is depicted in Figure 1.1.

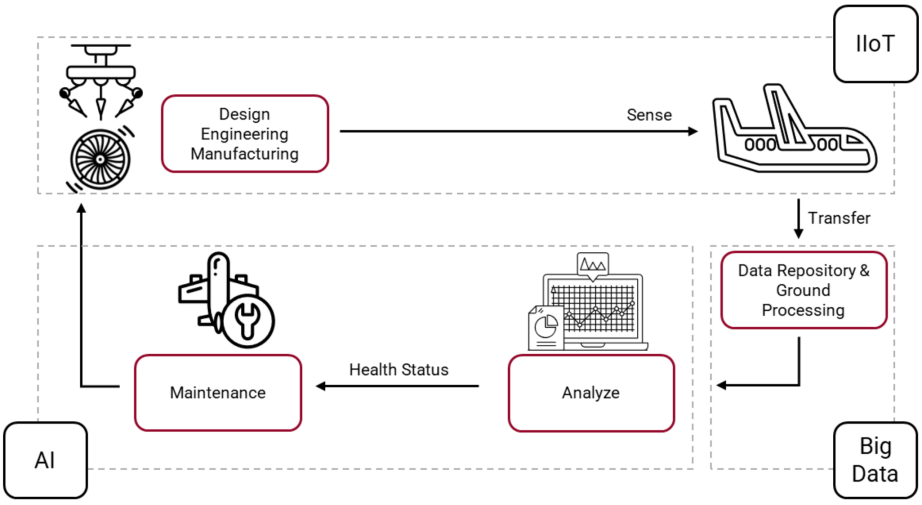


Figure 1.1: An Overview of Different Components in a Predictive Health Management System

1.1.2 Health Monitoring Systems

Machine health monitoring and management are essential improvements that must be considered in the physical industry toward smart manufacturing. Intelligent PHM systems are proving remarkable capabilities for industrial use and, consequently, have become active research areas in the last several decades [3]. The main benefits of implementing Integrated

System Health Monitoring (ISHM) are illustrated in Figure 1.2. ISHM refers to the procedure of gathering, examining, and utilizing data from various sensors and other sources in order to determine the current state of health of a system or piece of equipment. Using this data makes it possible to identify and assess prospective issues or failures before they arise and take the necessary corrective measures to avoid them. ISHM, prognostics, and other processes like maintenance planning, decision-making, and risk management are all included in the broader PHM notion. By providing real-time information on the condition of a system or piece of equipment, ISHM establishes the foundation for PHM, while prognostics uses this information to make predictions about the system's future health and Remaining Useful Life (RUL), which can be used to guide maintenance choices and enhance system performance.

Machines must be appropriately controlled and monitored to run with almost no breakdown and reduce the downtime caused by a local machine or component failure. Therefore, adequate sensors should be applied to devices to gather different measurements and monitor the components' health status in real time. Subsequently, the control center monitors and analyses the data gathered and then chooses and implements the best maintenance techniques for each machine and component [4]. Engineering maintenance is critical in various industries, including aircraft, manufacturing, automotive, and heavy industry.

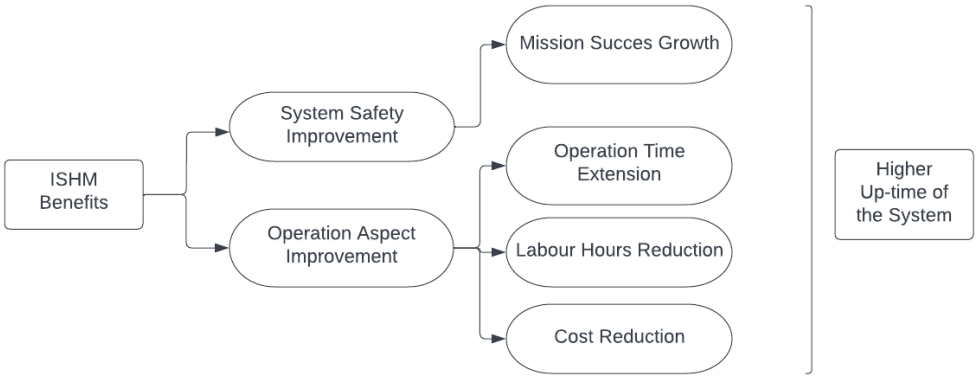


Figure 1.2: Benefits of Integrated System Health Monitoring (ISHM)

Modern industries gradually transform traditional diagnostic practices into prognostic procedures [5]. Diagnostics are examined to identify the nature of problems and faults. At the same time, prognostics are used to calculate and forecast the future status of machines or components by estimating their RUL until total breakdown [4]. Based on historical trajectory data, it is possible to predict the RUL, which is crucial for optimizing maintenance schedules to prevent engineering failures and save associated costs [3]. In general, prognostics monitors and detects early signs of decline in component performance.

1.1.3 Maintenance Strategies

Systems, machines, and their components have become more complex in recent years, notably for newer transportation vehicles such as airplanes, missiles, and military ships. Accordingly, a more complete and integrated system is required to manage total maintenance, operations, and decision-making aspects.

With the intention of PHM, numerous maintenance techniques have been suggested and put into practice for various asset types [6]. The traditional maintenance approaches, such as reactive maintenance (RM), condition-based maintenance (CBM), reliability-centered maintenance (RCM), and preventive maintenance (PM), as well as recently developed maintenance methodologies, such as maintenance based on information and communication technologies (ICT), self-maintenance, engineering immune system (EIS), and others, have aided in the development of PHM [4]. Another study [2] divided maintenance into two types: corrective (reactive) and preventive, while the latter consists of three sub-types: Interval-based Maintenance (IBM), Condition-based Maintenance (CBM), and Predictive Maintenance (PM).

1.1.3.1 Corrective Maintenance

According to [1], businesses may employ one or multiple maintenance strategies in their plants. 59% of them use a reactive maintenance strategy, meaning that corrective action is taken once a problem has occurred. As part of this run-to-failure approach, maintenance tasks are only carried out after a piece of equipment has malfunctioned. This approach can be expensive regarding missed production and repair expenses, even though it may be appropriate for non-critical equipment.

1.1.3.2 Preventive Maintenance

The most common kind of maintenance, preventive maintenance, is practiced by 76% of businesses [1]. In this technique, maintenance is typically carried out over short intervals regardless of the equipment's actual state [2, 7].

Interval-based maintenance (IBM), also referred to as time-based maintenance, is a maintenance technique in which maintenance tasks are carried out according to a regular timetable, regardless of the equipment's real state. Based on the presumption that the equipment will age at a given pace, maintenance tasks are scheduled to either delay or lessen the effects of failure.

Contrarily, condition-based maintenance (CBM) is a maintenance plan that specifies when maintenance tasks should be carried out on the actual condition of the equipment. When certain parameters are fulfilled, CBM uses real-time data, such as vibration, temperature, or pressure, to monitor the equipment's condition and initiate maintenance procedures. CBM aims to reduce wasteful maintenance, increase program efficiency overall, and carry out maintenance only when it is truly necessary.

Predictive maintenance (PM) has recently become popular for failure prediction [1] since it can forecast approaching failure and enhance the present maintenance system. For this reason, many companies have adopted this type of maintenance [1]. PM generally

predicts faults or breakdowns in a deteriorating system to optimize maintenance efforts by evaluating the system's status using historical data [6]. In this strategy, the RUL of the components is anticipated using characteristics, which typically include sensors, flight information, and operational profiles. By projecting necessary maintenance tasks for the future rather than only the present, predictive maintenance sets itself apart from CBM. Since failure is always foreseen, these strategies may be used to improve maintenance scheduling. Early predictive maintenance warnings may save costs by enhancing maintenance planning, decreasing downtime, and raising safety. Predictive maintenance can result in newer methods and better operations [3]. These strategies are briefly compared in Table 1.1.

Future failure monitoring makes it possible to use predictive maintenance procedures at the correct time, not too early when the equipment is still functional, or not too late when it has already failed. Predictive maintenance employs condition monitoring by utilizing various measurement technologies, including vibration analysis, infrared thermography, oil analysis, and acoustic monitoring, to track the equipment's real-time condition, identify impending failures, and proactively schedule maintenance actions.

Table 1.1: Maintenance Strategies

Strategy		Summary	Pros	Cons
Corrective Maintenance	Reactive Maintenance (RM)	Fixing when it breaks	Most accessible	Runaway repair costs
			Minimizing the involved costs before failure	High levels of machine downtime
		Run-to-failure strategy	The components are utilized to the max of their potential and life-span	High labor cost
			No database is required	Large stocks are required
Preventive Maintenance	Interval-based Maintenance (IBM)	Calculating a specific interval, based on failure history	Analysis can be done to determine when to replace a specific component	It is necessary to collect a large amount of failure data
				Parts may require inadequate repairs
		Widely usage in maintenance scheduling	The information may also be utilized to create a timetable for routine maintenance	Applicable to a minimal number of components
				Assuming that all of the components are evenly distributed and have the same failure distribution
	Condition-based Maintenance (CBM)	Analyses the behavior of a specific component or system using sensor data	Component health monitoring is possible before critical failure	When anomalies are detected, no maintenance recommendations are made
			The system may be monitored based on the component's direct condition	To monitor failure, numerous sensor and data solutions are needed
	Predictive Maintenance (PM)	Based on characteristics, the RUL of components is anticipated	Better maintenance scheduling	Higher initial investment

1.1.4 Applications of PHM

Prognostics and health monitoring systems could benefit many fields apart from engineering. Medicine [8], weather forecasting [9], military, batteries, structures [10, 11], and vehicles are examples of applications of prognostics in industry. In medicine, PHM has been applied to advise patients and healthcare professionals about the course of a disease and to estimate the patient's chance of survival. For example, in [12], a neural network was used to recognize the four distinct cancer growth patterns. PHM is also an essential element of weather forecasting since it enables meteorologists to forecast the probability of specific meteorological conditions at a particular time and location. Prognostics have been used in the military for a long time. However, specific applications are challenging to acquire due to security issues [13]. A significant amount of prognostic research has involved battery-powered applications. This type of system is critical for battery prognostics since batteries deteriorate over time. A dynamic model for battery health monitoring is presented in [14] and can be used to analyze structural component failure. However, battery deterioration is always present and has a distinct degradation pattern. This deterioration trend may be more challenging to identify when forecasting the RUL of engineering components. Finally, due to the rapid increase in the number of vehicles in the automotive field, protecting pedestrians and vehicle passengers has become an important objective for vehicle and vehicle parts companies, manufacturers, and official organizations responsible for verifying this measure. Prognostics have recently been implemented on vehicles for the sake of continuously evaluating diagnostics data over time to identify any significant potential degradation of vehicle subsystems that may cause a fault, predict the remaining useful life of the specific component or subsystem, and alert the driver before such a fault occurs [15]. The technique consists mainly of trending residuals taken from diagnostic data [13]. The metrics are mostly accuracy measurements such as mean squared error (MSE) or Gaussian probability density function (PDF) overlaps. The entire technique is data-driven and appropriate

when important baseline data is available [16, 17, 18].

One of the most crucial applications of PHM is aircraft maintenance since even minor equipment flaws can significantly affect how safely an aircraft operates. Worldwide air traffic is increasing, and operational performance must rise to meet the economy's demands. Optimum aircraft maintenance is necessary for high operational performance. Maintenance must be performed accurately to maximize uptime, better scheduling, and safety. Additionally, maintenance is estimated to account for 10 to 15 percent of all airline expenses [19]. Therefore, a more efficient failure prediction system can significantly benefit the industry [3]. System health assessments on spacecraft and airplanes are complex, expensive, and due to the unavailability of parts and components, sometimes impossible. Therefore, the aerospace sector is the focus of the most active research and development activity in systems prognostics. To decrease total lifespan costs and increase flight readiness, prognostic is being applied to the health management systems of the latest military and civilian aircraft. Governments, equipment manufacturers, small businesses, and even academia have conducted prognostics in aerospace [20]. Prognostic systems aid in maintenance scheduling, optimal operating mode determination, and asset purchase choices as they enter the commercial aircraft sector.

1.2 Problem Statement

Monitoring and maintenance have always existed. However, the demand for a comprehensive, integrated system for vehicle fault detection, diagnostics, failure prognostics, maintenance planning, operation decision support, and decision-making grew significantly with the fast development of engineering systems' complexity, particularly vehicles [21]. Maintenance management choices account for 15–40% of production costs across several industrial sectors, according to [2].

Any unexpected production interruptions are considered unplanned downtime. The

manufacturing line may experience a significant backlog due to the sudden stoppage, which might last for a while. Additionally, the cost is far higher in some industries, such as the auto sector, where downtime may cost up to 50 thousand dollars per minute or 3 million dollars per hour. On the other hand, a lack of sufficient PHM considerations has led to many catastrophic occurrences that have resulted in significant environmental harm and a significant loss of human life [22]. If appropriate maintenance practices had been followed, many disastrous incidents might have been avoided.

Existing PHM methods are classified into three types: model-based approaches, data-driven approaches, and hybrid approaches. While detailed modeling of the complex system degradation makes model-based techniques more accurate, it also necessitates substantial previous knowledge of physical systems, typically not accessible in practice, such as with aircraft engines [3].

The purpose of this research is to evaluate the possibility of predicting the RUL based on sensor data of a system by deploying artificial intelligence and, specifically, deep neural networks with an attention mechanism. First, the proposed solution needs to be feasible in terms of design, setup, and implementation. Then the predicted value for the RUL of one component or generally a system needs to be evaluated as a valid and verified estimation and at the same time practically applicable and acceptable.

1.3 Research Questions and Objectives

The main goal of this thesis is to design a predictor PHM model established on an attention-based neural network architecture to estimate the RUL of system components based on sensor data. We will provide support materials showing the proposed model's success. In addition, we will evaluate whether the proposed model is performing well by considering how late or early it can predict the failure of a component.

In order to achieve the goal above, the following sub-goals were achieved:

- Determining a system status according to its sensor data to see if it is operating in a normal healthy state or its degradation has started, see Section 3.4.
- Finding a way to detect and identify the most discriminating related sensor data for any system that does not need any knowledge from an expert from that specific domain, see Section 3.5.
- Evaluating the performance of the model and showing which metric is more suitable for this research domain, see Section 3.8.
- Determining the sensitivity of the model concerning quality and quantity of data, see Section 4.2.
- Evaluating the performance of attention-based models on time series data and comparing it with other architectures, see Section 4.3.

1.4 Organization of the Thesis

This thesis is organized into five chapters:

- Chapter 1: Introduction (current chapter)

  In this chapter, the background and motivations are introduced. The chapter starts with presenting smart manufacturing and health monitoring systems. Then, different maintenance strategies are described, and predictive maintenance is discussed in detail. Finally, the chapter presents the problem statement, followed by the goal and objectives of the research.

- Chapter 2: Literature Review and Theoretical Background

  In this chapter, we first review the required technical background. Then, the relevant literature on the topic for the following thesis developments is presented.

- Chapter 3: Problem Settings and Preprocessing

  This chapter describes the used datasets (C-MAPSS) [23], sensor details, sensor selection, preprocessing, operating conditions, and fault modes, showing the effects of different time window sizes and evaluation metrics.

- Chapter 4: Experiments

  In this chapter, we present the proposed model developed to meet the objective of this research. In addition, Chapter 4 includes the methodology and evaluation of the proposed model.

- Chapter 5: Conclusions and Future Perspectives

  This is the final chapter in which we summarize our work and wrap up the research by reviewing the limitations and future work possibilities.

Chapter 2

Literature Review and Theoretical Background

This chapter's main objective is to discuss previous works, fundamental definitions, and essential concepts required to understand this thesis. In Section 2.1, various facets of the remaining useful life (RUL) estimation problem are explained. Then, the state-of-the-art in the research area, presented in Section 2.2, gives an overview of the different models used in research and industry.

2.1 Background on Machine Learning

Machine learning methods have gained popularity recently due to their ability to handle large amounts of data and model complex relationships between input and output variables. Popular machine learning methods for RUL prediction include artificial neural networks and deep learning models for regression and line series analysis. These methods can be trained on historical data from the system or component to learn the degradation patterns and predict the RUL based on current or future operating conditions.

2.1.1 Machine Learning and RUL Prediction

RUL prediction involves predicting the time a particular component or system has before it fails or needs to be replaced. RUL prediction is an important application of machine learning in the field of predictive maintenance, where it is used to optimize maintenance schedules and reduce downtime. The basic idea behind RUL prediction is to train a model using historical data on the performance of the component or system. The model then uses this data to make predictions about the remaining useful life of the component, given its current state and usage. Figure 2.1 illustrates the concept of RUL graphically.

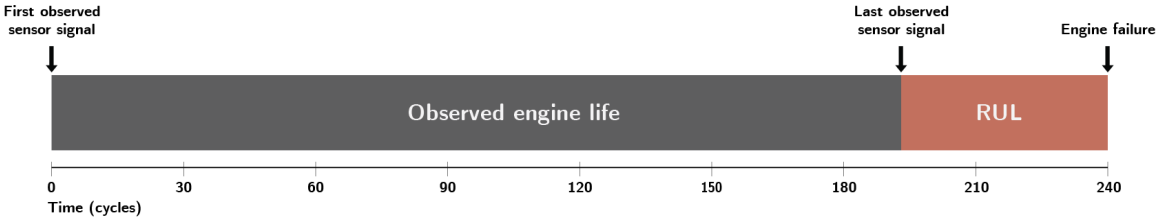


Figure 2.1: Depiction of RUL through Life Cycles of a Component

There are different approaches to RUL prediction, depending on the type of data being used and the structure of the model. One common approach is time series analysis, where the input data consists of time-varying signals such as sensor readings, vibration measurements, or acoustic emissions. These signals are used to model the degradation or wear of the component over time and to make predictions about its remaining useful life. Another approach is to use machine learning models such as neural networks, decision trees, or support vector machines for regression. These models can be trained on various features, including sensor data, environmental factors, and maintenance records. The goal is to identify patterns and relationships in the data that can be used to predict the remaining useful life of the component.

RUL prediction is a challenging task because it requires accurate models that can capture the complex and non-linear relationships between the input data and the remaining useful life of the component. It also requires careful data collection, preprocessing, and

domain expertise to interpret the results and optimize maintenance schedules. However, when done well, RUL prediction can significantly reduce maintenance costs, improve reliability, and increase uptime.

2.1.2 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) are machine learning algorithms loosely inspired by the structure and function of biological neurons and neural networks in the brain. ANNs recognize complex patterns and relationships in data by modeling the input data using a set of interconnected nodes organized in layers.

The basic unit of an ANN is a neuron, which takes a set of inputs, performs some processing on those inputs, and produces an output. The inputs to a neuron are multiplied by weights, and the sum of these weighted inputs is passed through an activation function to produce the output of the neuron. The activation function introduces non-linearity into the model, allowing ANNs to model complex relationships in data. ANNs are typically organized into layers of neurons, each consisting of a set of neurons connected to the neurons in the previous and next layers. The input layer receives the input data, while the output layer produces the final output of the model. The hidden layers perform intermediate processing and computation on the input data.

The weights of the connections between the neurons are learned from the data during the training process [24]. The objective of the training is to minimize a loss function that measures the error between the predicted output and the true output for each training example. This is typically done using an optimization algorithm that adjusts the weights to minimize the loss function. ANNs can be used for various tasks, including classification, regression, and clustering. They have been successfully applied in many areas, such as image and speech recognition, natural language processing, and financial forecasting. The effectiveness of ANNs in solving these tasks has made them one of the most widely used

machine learning algorithms today.

The architecture of an ANN can vary widely, depending on the specific task and the characteristics of the data. Some common types of ANNs include feedforward neural networks (FNN), recurrent neural networks (RNN), convolutional neural networks (CNN), and self-organizing maps. Overall, ANNs have proven to be powerful tools for solving complex problems in various domains. Their ability to automatically learn from data and model complex, non-linear relationships makes them a valuable tool for many applications in computer science and beyond [25].

Deep learning is a subset of machine learning which uses deep neural networks, i.e ANNs with many layers. Deep neural networks can learn hierarchical representations of data, allowing them to capture complex patterns and relationships in the data. Deep learning has been particularly successful in applications such as computer vision, natural language processing, and speech recognition, where large amounts of data are available, and highly complex relationships between the input and output variables exist.

One of the key advantages of deep learning is its ability to automatically learn feature representations from raw data, eliminating the need for manual feature engineering. This is often achieved through the use of convolutional layers, which are designed to learn spatial patterns in image data, and recurrent layers, which are designed to model temporal sequences in data such as text and speech.

Deep learning models are typically trained using backpropagation, a gradient-based optimization algorithm that adjusts the weights and biases of the network to minimize the difference between the predicted and actual output. The choice of optimization algorithm and hyperparameters, such as the learning rate and regularization strength, can significantly impact the model's performance.

While deep learning has achieved remarkable success in many areas, it has limitations. Deep learning models require large amounts of training data and are often computationally

expensive to train and deploy. They can also be prone to overfitting, where the model performs well on the training data but fails to generalize to new data.

Based on the ANN idea, several architectures were created, each with a unique design, information processing (data), and learning procedure. The most used approaches, which served as the cornerstones of deep learning are as follows:

1. Multilayer Perceptron Networks (MLPs)
2. Convolutional Neural Networks (CNNs)
3. Recurrent Neural Networks (RNNs)

We discuss and review the details of the first and the third structures as they have been deployed in our research work. Figure 2.2, depicts various neural networks and visually explains them.

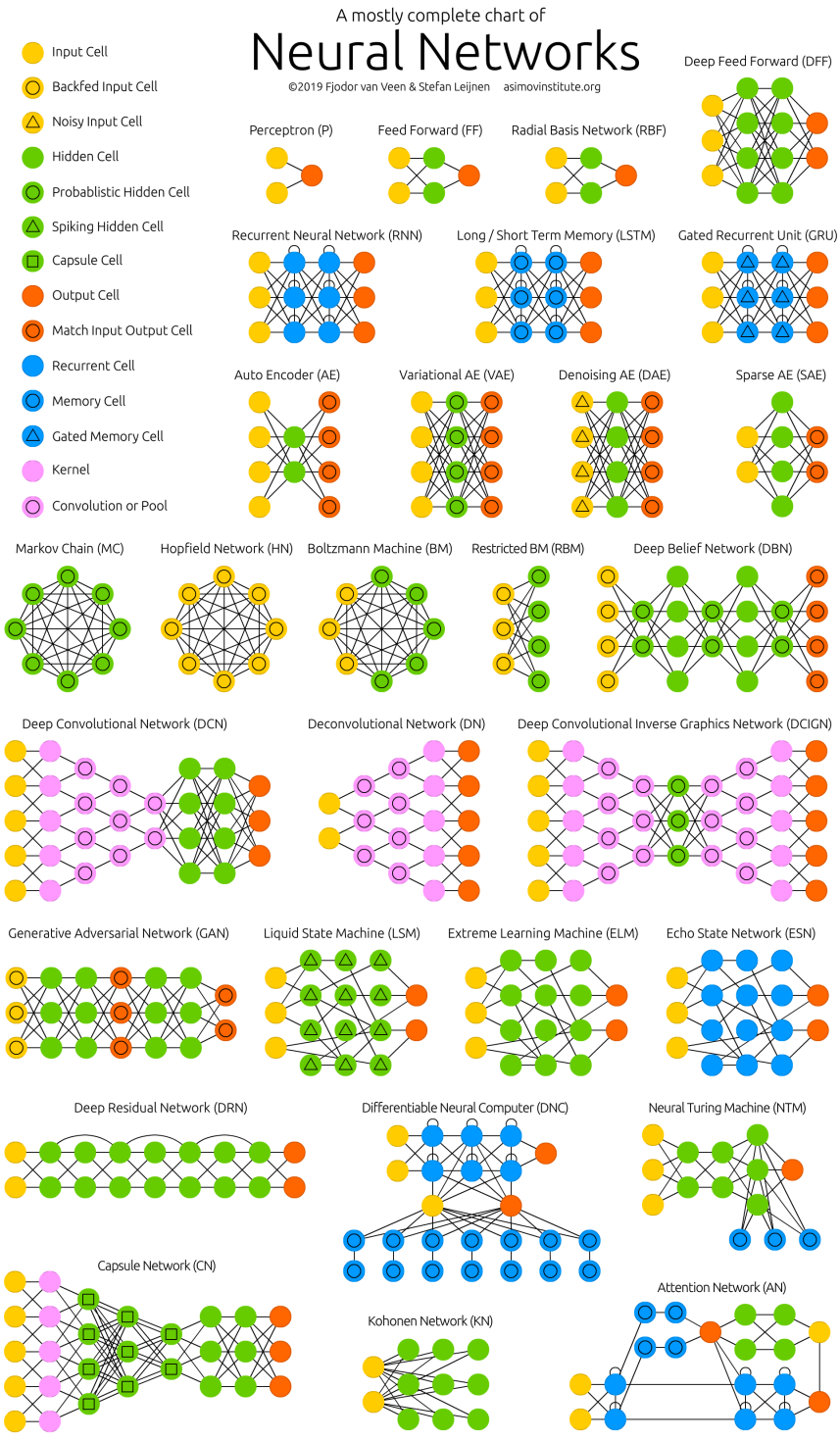


Figure 2.2: Neural Network Zoo by the Asimov Institute [26]

2.1.3 Multilayer Perceptron Networks (MLPs)

Multilayer perceptron (MLP) networks are feed forward artificial neural networks composed of multiple layers of perceptrons, which are the basic building blocks of ANNs [27]. MLP networks consist of an input layer, one or more hidden layers, and an output layer, as shown in Figure 2.3.

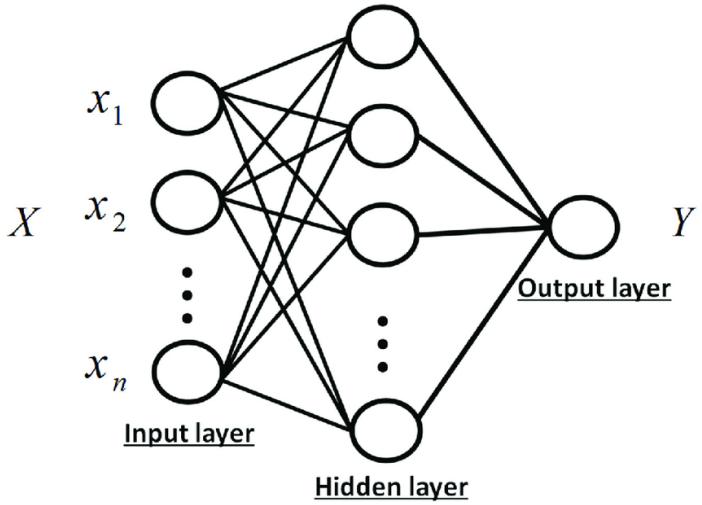


Figure 2.3: An Example MLP Network with One Hidden Layer [28]

The output of each layer is obtained by applying a nonlinear activation function to the weighted sum of the inputs. The weights and biases of the MLP network are learned through a process called backpropagation, which involves the propagation of error gradients from the output layer back to the input layer. The output y_k of a perceptron is calculated as Equation 2.1.

$$y_k = f\left(\sum_{j=1}^{n} w_{kj}x_j + b_k\right) \tag{2.1}$$

where x_j is the input to the j^{th} neuron, w_{kj} is the weight connecting the j^{th} input to the k^{th} neuron, b_k is the bias of the k^{th} neuron, f is the activation function, and n is the number of inputs to the neuron.

The most commonly used activation function for MLP networks is the sigmoid or logistic function, which has the form in Equation 2.2:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Other popular activation functions include the rectified linear unit (ReLU) in Equation 2.3,

$$f(x) = \max(0, x) \tag{2.3}$$

the hyperbolic tangent ($\tanh$) function shown in Equation 2.4,

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.4}$$

and the softmax function in Equation 2.5.

$$P_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.5}$$

where P_i is the probability of class i, z_i is the output of the i^{th} neuron in the output layer, and K is the number of classes.

The output of an MLP network is obtained by applying the activation function to the weighted sum of the inputs of the output layer (except for the softmax as it is defined above) as shown in Equation 2.6:

$$y_k = f\left(\sum_{j=1}^{n} w_{kj}^{(L)} y_j^{(L-1)} + b_k^{(L)}\right) \tag{2.6}$$

where L is the number of layers in the network, $w_{kj}^{(L)}$ is the weight connecting the j^{th} neuron in the $(L-1)^{th}$ layer to the k^{th} neuron in the L^{th} layer, $y_j^{(L-1)}$ is the output of the j^{th} neuron in the $(L-1)^{th}$ layer, $b_k^{(L)}$ is the bias of the k^{th} neuron in the L^{th} layer, and n is

the number of inputs to the L^{th} layer.

The weights and biases of an MLP network are learned by minimizing a cost function using an optimization algorithm such as gradient descent. The cost function measures the difference between the predicted output of the network (y_k), and the true output ($\hat{y}_k$) for a given input.

Overall, MLP networks are a powerful tool for modeling complex nonlinear relationships between input and output variables and have been successfully applied in many fields, including image and speech recognition, natural language processing, and financial forecasting.

2.1.4 Recurrent Neural Networks (RNNs)

To model temporal dependency in time series, recurrent neural networks (RNNs) with nodes connected along a sequence were developed [29]. Recurrent neural networks have been developed to handle sequential data. It consists of a chain of repeating modules of neural networks. This repeating module will have a very simple structure in standard RNNs, such as a single $tanh$ layer as shown in Figure 2.4.

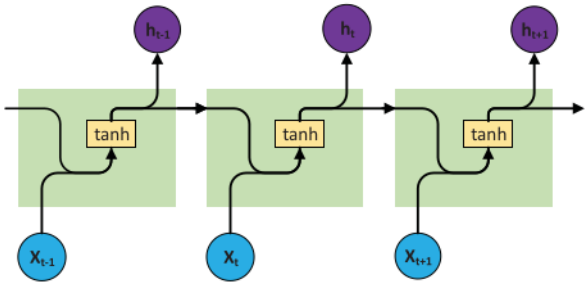


Figure 2.4: Three RNN Cells [29]

The main idea behind RNNs is to use the output of a module (or cell) as an input to the next module in the sequence, thus allowing the network to maintain a memory of previous inputs [30]. The formula for a simple RNN is:

$$h_t = f(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \tag{2.7}$$

$$y_t = g(W_{hy}h_t + b_y) \tag{2.8}$$

where h_t is the hidden state of the RNN at time step t, x_t is the input at time step t, y_t is the output at time step t, W_{hh}, W_{hx}, and W_{hy} are weight matrices, b_h and b_y are bias vectors, f is the activation function used for the hidden state, and g is the activation function used for the output.

The equation for the hidden state h_t shows that it is a function of the previous hidden state h_{t-1}, the current input x_t, and the network weights and biases. This allows the network to consider the previous inputs as it processes the current input.

One issue with this simple RNN is that the gradients can become very small (or large) when backpropagating through time. This is known as the vanishing gradient problem [30] and makes training the network difficult. When modeling long-term dependencies, the typical RNN performs significantly worse due to the issue of vanishing gradient or gradient exploding during network training [5]. The vanishing gradient problem arises because gradients become very small as they are backpropagated through many time steps, making it difficult for the model to learn long-term dependencies. Several variants of the RNN have been proposed to address this issue, such as the Long Short-Term Memory (LSTM) [31] and Gated Recurrent Unit (GRU) [32] networks.

2.1.5 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem that can occur when training standard RNNs on long sequences. An RNN is ideally suited for machine RUL prediction

using successive sensor readings. LSTM networks address vanishing gradient issues by introducing a memory cell and several gates to control the flow of information within the network.

The LSTM architecture, first introduced in [31], which may be thought of as a memory cell made up of a few gates, is a solution to this issue. The gates can capture long-term dependencies, which can permit or prohibit the passage of information along a sequence. Because of this, the LSTM network has had considerable success analyzing time-series data for tasks including occupancy prediction, video analysis, and natural language processing. Recently, it has also demonstrated excellent performance in terms of RUL prediction [29].

The memory cell of the LSTM is a vector that can retain information over long periods of time. The cell receives input from the previous hidden state and the current input and computes an updated cell state. The cell's output is then determined by a combination of the current cell state and the output gate, which controls how much of the cell state is used to compute the output. The input gate determines how much new input should be added to the cell state, and the forget gate controls how much of the previous cell state should be retained. These gates are learned during training, allowing the network to adapt.

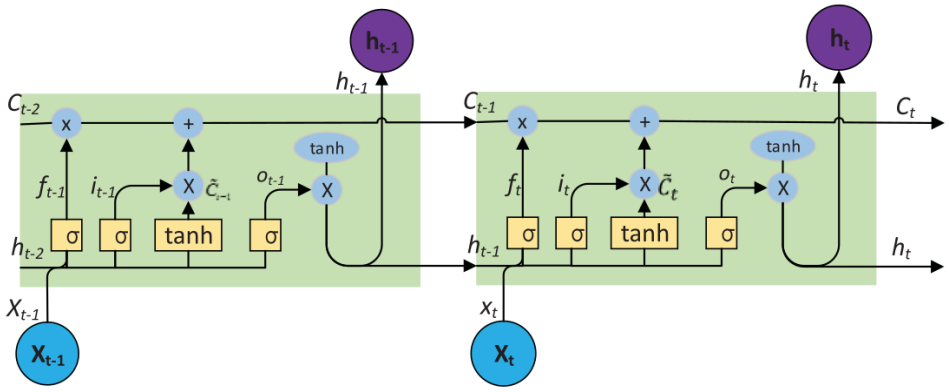


Figure 2.5: Structure of the LSTM [29]

In Figure 2.5, an LSTM network is depicted. An input gate chooses information from the input channel. A forget gate discards the unnecessary information from previous time

steps. An output gate is responsible for the outputs of the LSTM cell. Assuming that x^t is the input at time step t, h^t is the hidden state at time step t, C^{t-1} is the memory cell state, w^f, w^i, w^C, and wo^4 are the weights, b^f, b^i, b^C, and b^o are the biases, and σ and $\tanh$ are the $sigmoid$ and $\tanh$ functions, respectively, the LSTM network can be expressed as:

Forget gate:

$$f^t = \sigma(w^f[h^{t-1}, x^t] + b^f) \tag{2.9a}$$

Input gate:

$$i^t = \sigma(w^i[h^{t-1}, x^t] + b^i) \tag{2.9b}$$

Candidate cell state:

$$\tilde{C}^t = \tanh(w^c[h^{t-1}, x^t] + b^c) \tag{2.9c}$$

New cell state:

$$C^t = f^t * C^{t-1} + i^t * \tilde{C}^t \tag{2.9d}$$

Output gate:

$$o^t = \sigma(w^o[h^{t-1}, x^t] + b^o) \tag{2.9e}$$

New hidden value:

$$h^t = o^t * \tanh(C^t) \tag{2.9f}$$

2.1.6 Attention Mechanism

Applying attention mechanisms in neural networks helps models perform better on tasks that deal with sequential or structured data. It is especially useful when working with lengthy sequences since it enables the model to concentrate on the most crucial components of the input at each stage. The attention mechanism was first introduced in image processing to recognize objects and was based on the human visual system [33, 34]. Given

that humans often concentrate on a specific region of an image during identification, it makes sense that different portions of an image would be assigned variable weights. Attention has then been used effectively in works related to time-series data [35], and human language translation [36].

The fundamental idea underlying attention is to train the model to pay attention to various input components according to their relevance to the given task. This is accomplished by providing a series of learnable parameters that establish the relative importance of each input component [37]. A set of attention weights are computed for each input element or sequence of elements as part of the attention mechanism's high-level operation. The significance of each component for the current phase of the model is represented by these weights. The weighted sum of the input items is then calculated using the attention weights as the input to the model's subsequent step.

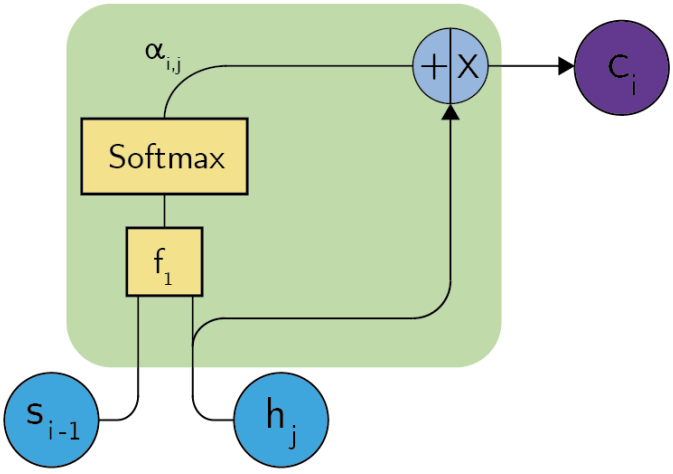


Figure 2.6: Structure of an Attention Module

Figure 2.6 shows an attention module in which the model tries to predict the attention weights ($\alpha_{i,j}$) according to Equation 2.10. Where $h_j \in \mathbb{R}^n$ is the original input with size n that the attention weight is calculated for it and $s_{i-1} \in \mathbb{R}^m$ is the hidden state with size m of the targeted value for each time step. f_1 is a linear or non-linear transformation that could be any neural network module. The vector $e_{i,j}$, also known as the alignment score,

measures the importance of j^{th} item of the i^{th} input sequence. This vector is normalized by softmax to give a vector of scores (α) indicating how much importance we should put on the i^{th} sequence.

$$e_{i,j} = f_1(s_{i-1}, h_j) \tag{2.10a}$$

$$\alpha_{i,j} = \text{softmax}(e_{i,j}) \tag{2.10b}$$

The next step is to apply the weights $\alpha_{i,j}$ to the original input elements h_j to get the final target values based on the attention weights. Equation 2.11 shows how the output c_i (context vector) is calculated. Note that the c_i is distinct at each time step i.

$$c_i = \sum_{j=0}^{n} \alpha_{i,j} h_j \tag{2.11}$$

Various attention mechanisms are deployed by deep learning models. Attention can be generally described as a weight or context vector of importance within a sequence [38]. In order to create a more accurate representation of a sequence, self-attention is a technique for focusing attention on the relationships between distinct positions within the sequence. Self-attention is a mechanism that relates different positions of a single sequence in order to gain a more explicit representation [39]. Instead of computing a complete summary context vector based on input/output prediction, as is the case with other attention mechanisms, self-attention instead calculates the relevance of each sequence token in relation to each other. To do this, a weighted total of each token in the sequence is calculated, with the weights acquired from the training. A function that evaluates each token in relation to every other token in the sequence determines the weight that should be given to it.

Although self-attention and attention function are similar in deep learning, there is a significant distinction between the two. When a model calculates attention scores between

various input components and another component of the input or external memory, the term "attention" is used. In machine translation, for instance, the attention mechanism computes attention scores between the source sentence and the target sentence, enabling the model to account for the relative weight of each component of the source sentence in the target translation. On the other hand, for self-attention, the model uses a mechanism to calculate the attention scores between various segments of the input sequence. Self-attention enables the model to assess the relative importance of each installment in the series, identify the interdependencies between them, and make predictions in light of that information.

In addition to introducing the self-attention mechanism, the work published as "Attention is all you need" [40] provided a very inclusive and expansive definition of the attention mechanism based on a key, a query, and a value. As shown in Figure 2.7, a query (Q) and a set of key-value (K, V) pairs are mapped to the output by an attention function, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, with the weight assigned to each value determined by the query's alignment score function with the corresponding key.

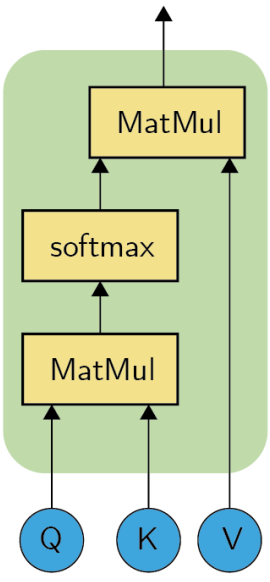


Figure 2.7: Self-attention Module in Generalized Form

According to Equation 2.12, given a query Q and a set of key-value pairs (K, V), it is possible to use attention in a more general way to calculate a weighted sum of the values based on the query and the associated keys. We can say that the query attends to the values by choosing which ones to emphasize. The self-attention mechanism can be thought of as a three-step sequence. First, it calculates the alignment score of the query and the keys, then applies the softmax function to convert the calculated score into a probability distribution, and finally, selects the key and calculates the output weights (context vector). All of the keys, values, and queries in a self-attention layer originate from the same source.

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V \tag{2.12}$$

where Q, K, and V are the corresponding query and key-value vectors and T is the transpose function.

There are several types of self-attention mechanisms deployed in different fields of deep learning. The main difference between these mechanisms is the alignment score calculation. Figure 2.8 shows the two most frequently deployed types, Scaled Dot-product Attention and Multi-head Attention.

- **Dot-product attention**: The dot product of the query and key is used to determine the alignment score. This was first introduced in [37].

- **Scaled Dot-product Attention**: The dot product between the query and key vectors is computed and then scaled by the square root of the dimensionality of the key vectors [40]. In transformer architecture, this is the type of self-attention that is employed the most.

  Scaled dot-product attention is a common and very powerful attention mechanism used in transformer-based models. The scaled dot-product attention output matrix is calculated in Equation 2.13.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.13}$$

where Q is the query matrix, which has dimensions $m \times d_k$, K is the key matrix, which has dimensions $n \times d_k$, V is the value matrix, which has dimensions $n \times d_v$, d_k is the dimension of the key and query vectors as they have a same dimension, d_v is the dimension of the value vectors.

- **Multi-head Attention**: The model can pay attention to various representations of the same input thanks to this particular form of self-attention. The process entails computing the alignment score for each head of the query, key, and value vectors before concatenating the results.

  Multi-head attention is a variant of the scaled dot-product attention mechanism that enables the model to simultaneously attend to information from different representation subspaces at different positions [40]. The multi-head attention formula is as stated in Equation 2.14.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{2.14}$$

  where Q, K, and V are the query, key, and value matrices. The number of attention heads denoted by h, and $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ is the output of the attention head i. Each attention head output is computed using the scaled dot-product attention mechanism of Equation 2.13. $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, and $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are learnable linear projection matrices for the query, key, and value inputs to the attention head i. The concatenated output of the attention heads is mapped to the output space of the model using a learnable linear projection matrix by $W^O \in \mathbb{R}^{h \times d_v \times d_{\text{model}}}$.

- **Relative Self-attention**: This type of self-attention computes attention by taking into account relative positional information. The query and key vectors are given learnable position embeddings to record their relative distance [41].

- **Local Self-attention**: This particular form of self-attention confines the calculation of attention to a small window of the input sequence. When modeling lengthy sequences, local self-attention can be helpful because the complete self-attention computation may be expensive [42].

- **Multidimensional Self-attention Block**: This type of self-attention is a sort of neural network layer that uses attention mechanisms to focus on relevant information from various dimensions or channels of input data. This model is employed mostly when the input is a multidimensional time series. It could be interesting to have one attention vector per dimension. A multidimensional attention block's core objective is to provide the model with the ability to selectively pay attention to various input data according to their importance for the current task. To do this, a set of attention weights are computed for each dimension or channel of the input data. These weights are then used to weigh the contribution of each element in that dimension to the layer's final output [43].

- **Sparse Self-attention**: This type of self-attention only considers a small number of input tokens while computing attention [44]. When simulating extremely long sequences when it would not be possible to do the whole self-attention computation, sparse self-attention can be helpful.

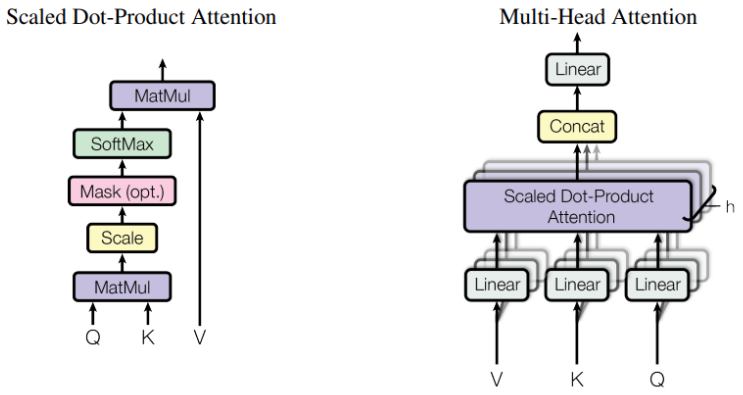


Figure 2.8: (Left) Scaled Dot-Product Attention. (Right) Multi-head Attention Consists of Several Attention Layers Running in Parallel [40]

Focusing on several areas of interest by allocating different weights to different attributes at various time steps, we suspect that will be an efficient method for machine RUL prediction. As we will see in Chapter 4, in this thesis, we will employ a self-attention mechanism to discover the relevance of the attributes and time phases as there is no prior knowledge indicating this relevance and links in the attributes.

2.1.7 Neural Network Settings and Hyperparameters

All neural network models include different parameters, such as model size (i.e. the number of hidden layers and neurons), type, learning method, activation function, etc. These are known as neural network settings and hyperparameters. The most important ones are explained in this section.

2.1.7.1 Neural Network Time Window

In a time-series neural network, a time window is a portion of the time series used to train the network at each time step. The time window determines how many prior time steps will be fed into the network to forecast the following time step. The time window size is a significant parameter impacting the network's performance. The network may not

have enough context to generate precise predictions if the time window is too limited. On the other hand, the network could become overloaded with information and have trouble learning if the time span is too large. The overlap across sequential time windows is an important parameter in addition to the time window's size. The network can perform better and capture long-term dependencies in the input by using overlapping time windows [45].

Grid search, random search, and Bayesian optimization are commonly used methods for choosing the optimal time window and overlap parameters. The aspects of the time series data, such as the size of the time series, the length of the time steps, and the complexity of the sequences and their correlations may affect the optimal values of these parameters.

2.1.7.2 Hyperparameters

Neural networks have many hyperparameters that can be tuned to improve their performance on a given task [46]. Here are some common hyperparameters and their descriptions.

- **Learning rate**: The size of the model's parameter updates during training is determined by the learning rate. Faster convergence can be achieved with a larger learning rate, but the optimization process could become unstable. Although a slower learning rate may lead to slower convergence and longer training cycles, it can help to assure stability.

- **Optimizer**: Neural network optimization is the process of changing a neural network's parameters to reduce the discrepancy between expected results and actual ones. Usually, an optimizer or optimization technique is used for this. Stochastic gradient descent (SGD), Adam, and RMS-prop are among the most well-known optimizers in neural networks.

- **Number of epochs**: The number of epochs indicates how often the neural network

is trained with the training data. The model's performance may improve with more epochs, but there may also be a greater risk of the model overfitting the training data.

- **Batch size**: How many training samples are utilized at a time to update the model's parameters depends on the batch size. Although a larger batch size might shorten training times, it might also use more memory and produce less consistent updates. Whilst it could take longer to train, a smaller batch size can help to ensure more stable updates.

- **Number of hidden layers**: The depth of the network and its capacity to learn complex patterns depend on the number of hidden layers in the neural network. Whereas a deeper network might be more capable of discovering complex patterns, it might also be more vulnerable to overfitting.

- **Number of hidden neurons**: The network's capacity and ability to express complex functions are determined by the number of hidden neurons. Better performance may come from using more hidden neurons, but overfitting is also possible.

- **Activation function**: The output of a node is determined by its activation function given an input or group of inputs. The activation function determines the non-linearity of the neural network and its ability to represent complex functions. Popular activation functions include ReLU, sigmoid, $\tanh$, and softmax.

- **Regularization**: The use of an appropriate regularization strategy helps to avoid overfitting. One of the most well-known solutions to this issue is a dropout. During training dropout, randomly sets a portion of the units in a layer to zero, hence blocking their contribution. The percentage of units that are arbitrarily dropped out at each iteration is determined by the dropout rate. It has been shown that the dropout regularization method works well in reducing overfitting in deep neural networks [47].

It can be used with several neural network architectures, including feedforward, convolutional, and recurrent neural networks. Dropout regularization, however, might potentially extend the model's training period because it needs to train multiple submodels.

Depending on the particular problem being solved and the properties of the data, the ideal hyperparameters for a given neural network will vary. Finding the ideal configuration for a particular task frequently requires testing various hyperparameter values and comparing their effectiveness on a validation set. Manual selection of the hyperparameters is possible, although not systematic. The majority of the hyperparameters interact with one another and provide a less-than-ideal model. Yet, there are methods [48] to find the optimal set of hyperparameters. For finding the optimal hyperparameters related to our proposed architecture we took advantage of the grid search technique.

Grid search involves giving a range of possible values for each hyperparameter and then evaluating the model's performance on a validation set for each hyperparameter combination. This can be a lengthy procedure, especially for models with a large number of hyperparameters or a wide range of possible values.

2.2 State-of-the-Art

Remaining Useful Life (RUL) is an important concept in prognostics, which refers to the amount of time a system has until it reaches a predefined failure threshold. Accurately predicting the RUL of a system can help to improve maintenance schedules, reduce downtime, and improve safety. There has been a growing body of literature on RUL prediction in recent years, focusing on developing accurate and robust prognostic models using various machine learning and statistical techniques. In this section, we discuss the literature review of recent research on RUL prediction.

The main methods for forecasting RUL in the literature may be classified into physical model-based techniques, data-driven model-based approaches, and ensemble-based & hybrid methods [49]. We review each category and discuss recent research in these categories.

2.2.1 Physics-based Techniques

Physics-based techniques characterize a system's deterioration stage by building mathematical models based on failure mechanisms or the first principle of damage [50]. The physical model developed with a thorough understanding of failure mechanisms and excellent parameter estimates may give reliable RUL estimation. Physics-based models consider the physical degradation mechanisms of the system and can provide more interpretable predictions. Physics-based techniques are the most used in the industry as they are extremely trustworthy once the model is developed [51].

However, physics-based techniques have two major drawbacks. First, the developed physical model is difficult to apply directly to other systems. Due to their dependence on the behavior of the particular industrial system under study, the solutions in this category only apply to that system. They are not readily transferable to other systems. In addition, they need in-depth experimentation, specialized knowledge, and model verification, which may be difficult and occasionally impossible for complex systems. They need a strong grasp of the physical process of failure. Establishing an efficient physical model necessitates sophisticated previous information. Because of these constraints, data-driven techniques are becoming more popular today.

2.2.2 Data-driven Approaches

Data-driven approaches to RUL prediction have gained significant attention in recent years as they do not require explicitly modeling the system's underlying physics. They use trained models based on historical data from sensors integrated into manufacturing systems

to model the degradation characteristics. Machine learning, specifically deep learning, has been widely applied in RUL prediction due to its ability to learn complex patterns and features automatically. For example, [52] proposed a support vector regression (SVR) model for predicting the RUL of rolling bearings. The model uses a set of features extracted from vibration signals to train an SVR model to predict the remaining useful life of the bearing. These models can access various data formats and take advantage of data variances that physics-based techniques cannot recognize. [53] proposed a deep belief network feedforward neural network (DBN-FNN). Using the help of the learned features, the FNN performed the RUL prediction after learning representative features with the DBN.

A deep convolutional neural network (CNN) is suggested by [54] for the prediction of RUL. They investigated the algorithm's empirical performance on two public datasets, the NASA Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) and the PHM 2008 Data Challenge data set, as the first attempt to use CNN for RUL estimation in prognostics. The results demonstrate that the suggested CNN-based RUL estimation method outperforms current regression methods. The experimental findings on two datasets demonstrated that the CNN outperformed several shallow learning algorithms for RUL prediction [55] and implemented a multiscale convolutional neural network (MSCNN) for RUL prediction. First, raw sensory data was subjected to a wavelet transform for a time-frequency representation (TFR). The TFR was then utilized as an input to the MSCNN for RUL prediction. [56] explored the time-frequently domain information for prognostics and then suggested an RUL prediction method. They used convolutional neural networks on a well-known rolling beating dataset to extract the multiscale feature. The outcomes of the experiments indicate that the suggested approach holds promise for prognostic issues and is well-suited for commercial use.

A long short-term memory (LSTM)-based RUL prediction is presented in [57]. The

experimental findings on three datasets revealed that the LSTM outperforms several shallow learning techniques like CNN [58] suggested a bidirectional long short-term memory (Bi-LSTM) approach for RUL prediction. A health index (HI) is first introduced, then the Bi-LSTM is used to track the final RUL forecasting index variation. The RUL prediction uses a multi-objective deep belief network ensemble (MODBNE) approach from [59]. They used a multi-objective evolutionary method to train DBNs with two competing goals: accuracy and variety. The developed DBNs were then pooled to build an ensemble model for the final RUL prediction.

One common deep learning-based technique is to characterize RUL estimation as a regression problem, directly translating input data to RULs. [58] used competitive learning to divide the training data into multiple degradation phases. Then, using historical data before the latest state, an Elman RNN network was trained. Finally, the performance of the most recent state was forecasted using this trained RNN. An LSTM network is applied by [60] and [61], for directly estimating the RUL. A recent study further attaches a restricted Boltzmann machine (RBM) before the LSTM layer. This unsupervised pre-training stage is used to learn abstract characteristics from raw unlabeled input data automatically. Furthermore, a genetic algorithm (GA) was used to adjust a large number of hyper-parameters [62].

Autoencoders have also been used to extract features from high-dimensional sensor data for RUL prediction. [63] proposed a model based on stacked denoising autoencoders for predicting the RUL of bearings. The model uses a set of features extracted from vibration signals to train the autoencoder model to predict the remaining useful life of the bearing. An ultimate intelligent technique for prognosticating and monitoring the condition of standard rotary machine bearings was proposed in [64]. Their approach was based on autoencoder-based unsupervised feature extraction from sensors. A valuable trend on the state of the bearing during the test-to-failure was produced as a result of a correlation analysis on the

retrieved features. It was determined that autoencoders accurately predicted the degradation's starting point while providing an informative trend on how it progressed. It also successfully monitored the health of the bearings in several experiments. An attention-based LSTM is also used in the model proposed in [65] alongside a time series multiple channel convolutional neural network (TSMC-CNN) to predict the RUL for bearings. The general attention mechanism is used for decoding the extracted features as the context vector by the CNN model and feeding the features with different weights to the LSTM network. The approach showed significant improvement in performance in comparison to the architecture without taking advantage of the attention layer before the LSTM.

As described above, data-driven approaches have been applied successfully in RUL prediction and have shown promising results in various applications. These approaches provide a more practical alternative to physics-based models, which require extensive knowledge of the underlying degradation mechanisms of the system. Accurate and reliable RUL prediction models are essential for improving maintenance schedules, reducing downtime, and improving the safety of critical systems.

2.2.3 Hybrid and Ensemble-based Methods

Ensemble methods combine multiple models to improve the performance of RUL prediction. Hybrid models combine both data-driven and physics-based approaches to RUL prediction. [66] proposed a hybrid model based on SVR and a degradation law for predicting the RUL of a gas turbine, achieving high accuracy compared to other models. In another research [67], high accuracy on prediction is observed by a hybrid CNN and LSTM model to predict RUL in a bearing dataset, compared to other models. In another study, [68] proposed an ensemble model based on multiple regression models for predicting the RUL of a gearbox, achieving better performance than individual models.

In conclusion, RUL prediction is an active research area, and various machine learning

and statistical techniques have been applied to this problem. Accurate and robust RUL prediction models can help to improve maintenance and reduce downtime, leading to significant benefits in many industrial applications. Table 2.1 summarizes the state-of-the-art approaches to RUL prediction presented in this section.

Table 2.1: State-of-the-art for RUL Prediction

Approach	**Year**	**Dataset**
Autoencoder [64]	2017	Bearing
CNN [56]	2018	Bearing
DBN [69]	2019	Bearing
TSMC-CNN-ALSTM [65]	2020	Bearing
CNN [54]	2016	Turbofan
Multi-DBN [59]	2016	Turbofan
LSTM [57]	2017	Turbofan
LSTM + CNN [70]	2019	Turbofan
Autoencoder [71]	2019	Turbofan
Optimized LSTM-GRU [72]	2021	Turbofan
LSTM [73]	2020	Battery
Bi-LSTM [74]	2020	Battery
CNN-LSTM [67]	2021	Battery
MFO-DNN [75]	2021	Battery

2.2.4 Literature Gap

Due to the limitations of generic RNNs on long-term dependencies, due to the vanishing gradient problem, these methods do not perform very well on RUL prediction tasks, especially ones having datasets with longer sequences. On the other hand, although GRUs

are very fast to train and less computationally expensive than LSTMs, LSTMs often GRUs outperform when dealing with long-term dependencies and datasets with longer sequences.

The LSTM network, developed to handle sequential data modeling, is naturally suited for RUL prediction because the sensory data for PHM are time series with temporal dependence. The different features the LSTM network learns at various time steps will contribute equally to the RUL's final prediction, which is not necessarily representative of one problem. Giving more weight to features and time steps that are more significant may be more representative. As a result, to improve the performance of the RUL prediction, we suggest a self-attention-based deep learning method alongside an LSTM to automatically give greater weights to more important features and time steps on the input data and the set of learned features.

2.3 Chapter Summary

This chapter mainly investigated the field's previous works and reviewed the fundamental definitions and essential concepts required for our research. Section 2.1 reviewed work on RUL prediction and how machine learning has been used to help in predictive maintenance. The section explained the fundamentals of artificial neural networks. We targeted recurrent neural networks and investigated their structure, advantages, and disadvantages. We explained that LSTM networks were specifically introduced to address the vanishing gradient problem which RNNs suffer from. Then, we covered the attention mechanism that helps neural networks to perform well on tasks that model sequential or structured data. By identifying the most crucial components of features, this mechanism helps work with even lengthy sequences. We reviewed different attention mechanisms and went through their implementations. Finally, we discussed the most important hyperparameters in neural networks.

In Section 2.2, we reviewed the approaches and different models proposed in academic

research and in the industry for prognostics. We first reviewed some works on physics-based models and pointed out their advantages and disadvantages. Then, we targeted data-driven approaches and investigated recent advancements and research trends in the field. Next, we reviewed research works done on hybrid and ensemble-based models, which exist in the field that we will address in this thesis. These models combine multiple models to improve performance. Finally, we summarized state-of-the-art works done in recent years for prognostics and predictive maintenance with their corresponding targeted industry and explained the gap which we think exists in the field.

In the next chapter, we will explain the experimental setup, the dataset used in this research, and preprocessing steps for the data preparation to feed to our proposed network. We will investigate the data environment in detail and then, according to our observations, will describe the procedures we used to prepare the data for our experiment.

Chapter 3

Experimental Setup and Preprocessing

The goal of this chapter is to describe the dataset used in our research and its features, and then explain the necessary steps to prepare the dataset to be fed to our proposed model described in Chapter 4. This chapter is structured as follows. In Section 3.1, we first introduce the dataset and describe it in detail. Second, in Section 3.2, the data from 21 different sensors, plus 3 operation conditions are reviewed. In the next session, we go through preprocessing and parameter adjustment for the training model step. Finally, metrics and target functions for remaining useful life (RUL) are discussed in Section 3.8.

3.1 Benchmark Dataset

The Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)[23] dataset has been used as a dataset in this work to train and evaluate the effectiveness of our proposed model for RUL prediction. C-MAPSS is a simulation tool coded in the MATLAB-Simulink environment for simulating the engine model of the 90,000 lb thrust class [76]. The simulation tool was used to generate a benchmark dataset, known as the C-MAPSS dataset. Although the dataset was generated nearly 15 years ago, it remains one of the most well-known and used datasets in fault detection, prognostics, and diagnostics studies

today [20]. One reason for this is the unavailability of other datasets in the field that can provide the same level of standard in terms of quality. The C-MAPSS dataset was created to detail the degradation process of aircraft turbofan engines. A simplified depiction of that is shown in Figure 3.1. The data is simulated in a way to show the fault impact and the degradation of the main components of the turbofan engine. A few important components of the engine include a Fan, a Low-Pressure Compressor (LPC), a High-Pressure Compressor (HPC), a Low-Pressure Turbine (LPT), and a High-Pressure Turbine (HPT), which are illustrated in Figure 3.2.

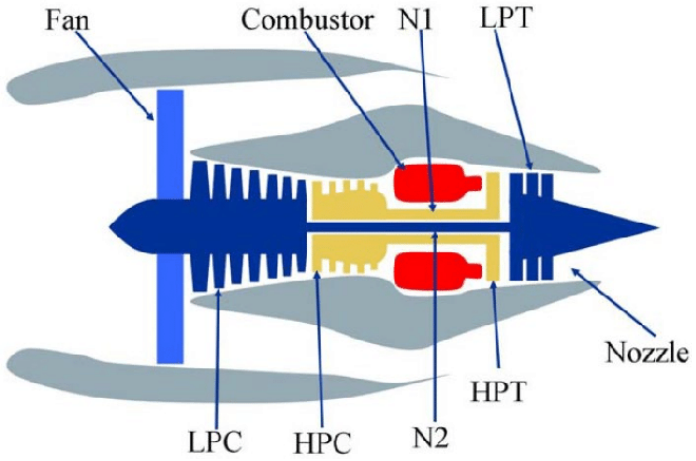


Figure 3.1: Simplified Diagram of Engine Simulated in C-MAPSS [76]

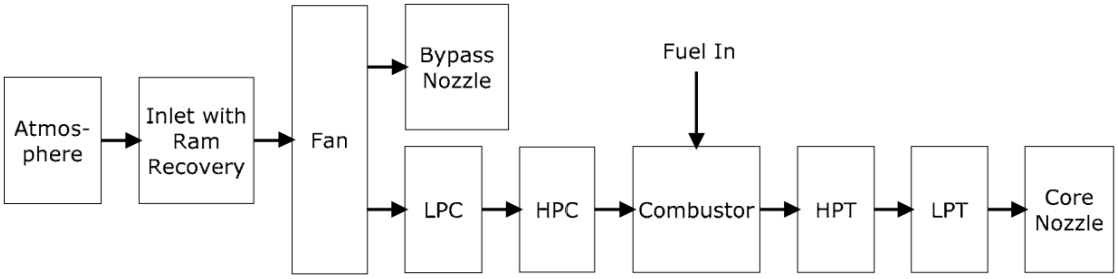


Figure 3.2: Layout of the Relation between the Main Modules as Modeled in the Simulation of C-MAPSS [76]

The dataset is divided into four sub-datasets (referred to as FD001 to FD004) by considering different fault modes and operation conditions. These are shown in Table 3.1. Each

sub-dataset also has a unique subset of multiple training and test sequences. These sub-datasets include time-series measurements of different sensors inside one turbofan engine. All engines are of the same product type, but on their first cycle (first set of measurements), they do not have the same level of degradation. The status for the early stages of each engine, though, is healthy and functional. With the progression of time, the degradation impact on the engines becomes increasingly important and eventually leads to engine breakdown. On the training sequences, for each engine unit, engine breakdown status is reported on the last data entry, while on the test sequences, the last data point for one specific engine corresponds to one particular point before engine breakdown. The objective of the test dataset is to estimate the RUL. For each engine in the test sequences, the actual value of the RUL is reported in the C-MAPSS dataset to validate the model performance.

Table 3.1: CMAPSS Dataset Overview

Sub-Datasets	**Training Units**	**Testing Units**	**Operation Conditions**	**Fault Modes**
FD001	100	100	1 (Sea Level)	1 (HPC)
FD002	260	259	6	1 (HPC)
FD003	100	100	1 (Sea Level)	2 (HPC, Fan)
FD004	249	248	6	2 (HPC, Fan)

Operation conditions are set based on the values of the 3 operating conditions settings. These are altitude, Mach number, and the throttle resolver angle in which the simulation takes place [23]. Table 3.2 shows these settings alongside their range and unit.

Table 3.2: CMAPSS Operating Conditions Settings

Operating Settings	Range	Units
Altitude	$[0, 42000]$	feet
Mach Number	$[0, 0.84]$	-
Throttle Resolver Angle	$[80, 100]$	$^\circ$ R

3.2 Notations for Time-series Data

In order to facilitate the reading of the rest of the thesis, we provide here the mathematical notation used in our experiments.

- $N \in Z$ is the total number of samples in the dataset. We may also refer to N, as the total number of observed samples or dataset size. In the case of our dataset, N varies between 100 and 249 for FD001 and FD004 respectively.

- $M \in Z$ is the total number of features. We may also refer to M as the dimension or the number of attributes. In our experiments $M = 24$. We have 21 sensors plus 3 operation conditions that act as features.

- $P \in Z$ is the total number of samples in a series. We may also refer to P as the length or the size of samples in a series. In our work, P is between 128 and 362 on FD001 and 128 and 543 on FD004.

- $X = \{x_0, x_1, x_2, ..., x_{M-1}\}$ is a set of M features. We may also refer to X as a set of input values, sets of attributes, or parameters. Each $x_i \in X$, $i \in [0, M)$, belongs to a class of possible values. These values may be uninformative, nominal, ordinal, binary, continuous, or discrete.

- Useless: denotes non-related features whose value will not change the outcome value. For example, a unique identification number is a useless feature, and in changing its value, there is no functional effect on the outcome. If our dataset includes a feature with such characteristics, there is no point in involving it in our model. In our dataset, Engine Id belongs to this class of values. It acts as an identity key for determining a particular engine in the dataset.
- Nominal (or Categorical): consists of discrete values without a meaningful numerical relationship between values. Applying most numerical functions to them is out of consideration. There is no example of this data type in our dataset.
- Ordinal: denotes discrete integer values that can be sorted. The main characteristic of this type of value is that there is no fixed distance between two values of this class of data. In our dataset, RUL is from this type of data.
- Binary: A binary data can only accept two values. This type of data is very common as a result or outcome variable in classification. For example, failure is a flag that only has two possible values, whether it has failed or not.
- Continuous: This type of data accepts values over an existing range of real numbers. All uncountable and infinite values are part of this data type. An example of a continuous variable in our dataset is the mass flow rate of air entering an engine.
- Discrete: Discrete variables take countable integer values, and in comparison to ordinal and binary variables, all statistical methods can be applied to them. For example, the number of cycles to failure is a discrete variable. This data type also consists of interval variables and time series. In our dataset, operation conditions belong to this type of data.

- $Y = \{y_0, y_1, ...\}$ with its corresponding domain represents the outcome variable, target, or label. The domain, according to the problem, is determined. For regression, a number (continuous or discrete) acts as the label for the machine learning algorithm, while for classification, the label value will be in a binary or categorical format. For example for continuous RUL, we have $y_{s_i} \in [0, max(RUL)]$, for binary type failure we have $y_{s_i} \in \{0, 1\}$, and for categorical RUL, the label should be $y_{s_i} \in \{RUL_0, RUL_1, RUL_2, ..., RUL_Q\}$ where $Q \in Z$ is the total number of discrete values for RUL. For the purpose of this study, we consider the domain of label (RUL) as a numerical type that will be used in the regression problem.

- $S = \{s_0, s_1, s_2, ..., s_{N-1}\}$ is a set of N samples. Each $s_i \in S$, $i \in [0, N)$, has M features $x_{s_i} = \{x_{(s_i,0)}, x_{(s_i,1)}, x_{(s_i,2)}, ..., x_{(s_i,M)}\}$, plus a label y_{s_i}.

$$\begin{bmatrix} x_{(s_0,0)} & x_{(s_0,1)} & x_{(s_0,2)} & \cdots & x_{(s_0,M-1)} \\ x_{(s_1,0)} & x_{(s_1,1)} & x_{(s_1,2)} & \cdots & x_{(s_1,M-1)} \\ x_{(s_2,0)} & x_{(s_2,1)} & x_{(s_2,2)} & \cdots & x_{(s_2,M-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{(s_{N-1},0)} & x_{(s_{N-1},1)} & x_{(s_{N-1},2)} & \cdots & x_{(s_{N-1},M-1)} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{bmatrix} \tag{3.1}$$

- $T = \{t_0, t_1, t_2, ..., t_{P-1}\}$ is a set of time series. For each $t_i \in T$, $i \in [0, P)$, we have $t_i = \{s_{(t_i,0)}, s_{(t_i,1)}, s_{(t_i,2)}, ..., s_{(t_i,P-1)}\}$. $Max(RUL)$ is the maximum value of RUL in the all-time series. $RUL_{t_{(s_i,j)}}$ is the RUL of sample s_i, $i \in [0, N)$ for the j^{th}, $j \in [0, P)$, time series in T.

3.3 Conditions and Sensors

As stated in Section 3.1, based on the values of the operating conditions and fault modes, the C-MAPSS dataset is divided into four sub-datasets. By taking operating conditions

into account, firstly, the sub-datasets are divided into two groups. As shown in Table 3.1, the first group includes FD001 and FD003, with only a single condition of operating at sea level for the simulation. The second group includes FD002 and FD004, which are simulated under the impact of six conditions. The next dataset division is done based on the modes for the failure. This also groups the sub-datasets into two. The first group includes FD001 and FD002, which have HPC as their only fault factor, and FD003 and FD004, which have fans alongside the HPC as their fault factors [77].

According to this simulation setup, the least complex sequences are in FD001, as this dataset only has a single fault mode and single operating condition. On the other hand, the most complex sequences are in FD004, which has two fault modes and six different operating conditions. As FD001 is the least complex and FD004 is the most complex dataset to cover different complexities, we will use these two sub-datasets for training and evaluating our proposed model. Due to the fact that FD002 and FD003 are special cases of FD004 and FD001 respectively, the results on FD001 and FD004 can be generalized on the other two to a certain extent. Also for most of the experiments and preprocessing steps on the C-MAPSS dataset, we found quite similar behavior from both FD001 and FD004 datasets. So, in the rest of the thesis, we will only illustrate FD004 separately if we find any different trends or behavior on that dataset at any particular step.

Each sub-datasets is represented by a $N \times 26$ matrix, where N is the number of total time cycles with 26 columns of numbers as their attributes. The first column is the engine unit index, and the second column is the sequence time cycle index. Then, we have three attributes for the operating conditions settings that are discussed above. These settings as discussed in Section 3.1, are the values of the altitude, Mach number, and throttle resolver angle. The next 21 columns are sensor signals from various components of the aircraft turbofan engine. These sensor signals, alongside a short description and measurement units, are shown in Table 3.3. The complete details of the sensor data can be found in [23].

Table 3.3: Sensor Description of the C-MAPSS Dataset

Sensor ID	Symbol	Description	Units	Trend
S_1	T2	Total Temperature at fan inlet	° R	∼
S_2	T24	Total temperature at LPC outlet	° R	↑
S_3	T30	Total temperature at HPC outlet	° R	↑
S_4	T50	Total temperature LPT outlet	° R	↑
S_5	P2	Pressure at fan inlet	psia	∼
S_6	P15	Total pressure in bypass-duct	psia	∼
S_7	P30	Total pressure at HPC outlet	psia	↓
S_8	Nf	Physical fan speed	rpm	↑
S_9	Nc	Physical core speed	rpm	↑
S_10	Epr	Engine pressure ratio	-	∼
S_11	Ps30	Static pressure at HPC outlet	psia	↑
S_12	Phi	Ratio of fuel flow to Ps30	pps/psi	↓
S_13	NRf	Corrected fan speed	rpm	↑
S_14	NRc	Corrected core speed	rpm	↓
S_15	BPR	Bypass ratio	-	↑
S_16	farB	Burner fuel-air ratio	-	∼
S_17	htBleed	Bleed enthalpy	-	↑
S_18	NF dmd	Demanded fan speed	rpm	∼
S_19	PCNR dmd	Demanded corrected fan speed	rpm	∼
S_20	W31	HPT coolant bleed	lbm/s	↓
S_21	W32	LPT coolant bleed	lbm/s	↓

3.4 Labeling and RUL Target Function

As any supervised machine learning model needs a target variable to do the prediction task, and since RUL prediction is a typical regression problem, we also need to generate labels for the training data to be used as the target variable for our proposed regression model. To achieve this, we use RUL as the output for our sensor signals input data. Figure 3.3 shows different stages of an engine's health status through its life cycles from starting point to failure. It also illustrates how RUL is determined according to the time step.

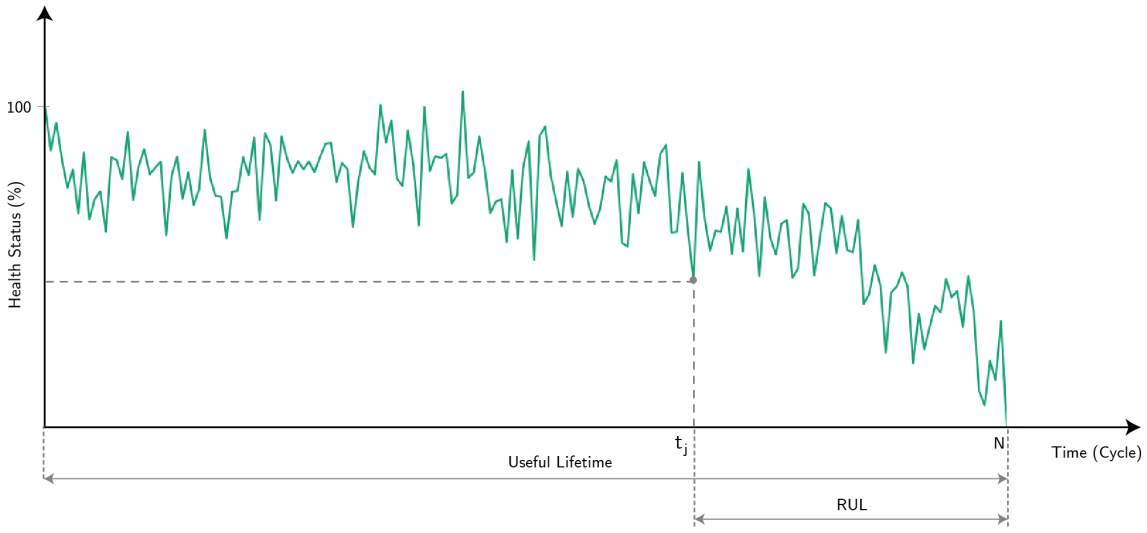


Figure 3.3: Depiction of an Engine Health Status Through Time

Because the RUL value is not available explicitly in the C-MAPSS dataset, we need to first calculate the RUL for each engine unit. In order to calculate RUL and label the dataset, in this thesis, we first defined the value of RUL based on Equation 3.2a, where i is the engine unit index, j is the current engine time cycle, and N_i is the maximum value for the time cycle for the i^{th} engine. The $RUL_i(j)$ is the remaining useful life of i^{th} engine at time cycle j. By definition, the maximum value for RUL for each engine unit is the value of N_i, shown in Equation 3.2b. This indicates that the value for the RUL in one engine has its maximum on the first sequence ($j = 0$), and it reaches 0 ($j = N_i$) on the last sequence.

$$RUL_i(j) = N_i - t_j \tag{3.2a}$$

$$\max(RUL_i) = N_i \tag{3.2b}$$

The RUL value is used for two purposes. First, it is deployed alongside sensor signals to give us a better interpretation of the changes in the sensor signal as the engines move toward failure. The second purpose is to label the datasets, allowing us to predict engine failure in our proposed model. So in other words, we first calculate RUL to label the C-MAPSS dataset and use it as the actual remaining life of the engine units, and then based on the sensor data trend we try to predict the value of RUL for each unit on each time step.

Basically, it is best to have precise knowledge about the RUL value (label here) for each input data point to increase performance. However, this information is not available in PHM problems due to the lack of an accurate physics-based model, so it is common practice to try to estimate the health status of the system in real applications.

Using the definition for RUL in Equation 3.2a, we consider the degradation process as a declining linear trend on the RUL on each engine time cycle proposed in [78]. According to the definition of the remaining useful life of a component, a declining linear trend is the most straightforward and, at the same time, the most rigorous way to show the natural degradation of a component against the passing of time when there is no knowledge or insight available to consider, although a declining linear trend seems simple when we compare it to the sensor signals trends.

Figure 3.4 shows the signal trends of an arbitrary sensor (S_7) and compares it to a linear degradation RUL (in blue) and a piece-wise RUL (in orange), first used in [79], that divides the RUL into a healthy initial stage and a degradation stage. As the figure shows, the trends are rather constant and straight at the beginning of the engine lifetime, because

the degradation process in a system, generally, develops over time and appears at a certain point of the component lifetime. This behavior was observed on some sensors in the dataset that will be discussed in Section 3.5 in detail. Hence, we used the piece-wise target function of [79].

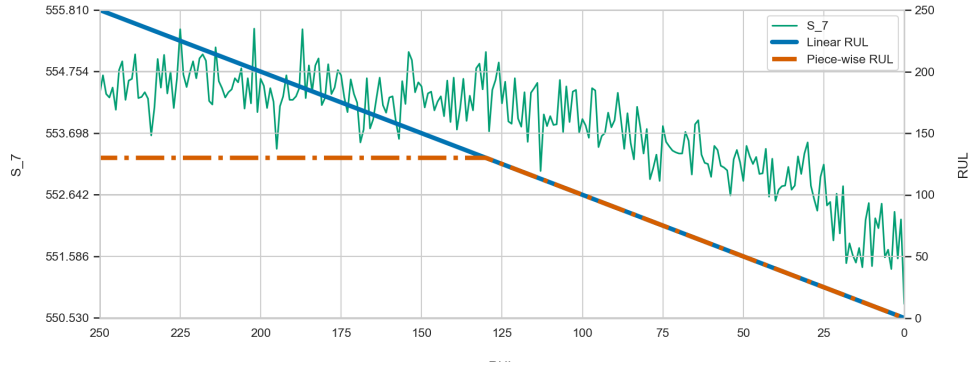
As long as the system is in the healthy stage, there is no visible trend or behavior on sensor signals indicating any degradation or abnormality. This RUL target function deploys the maximum RUL defined as RUL_{max} according to each sub-dataset sensor signals trend and sets the numerical value for the RUL above that to the maximum RUL as defined in Equation 3.3, where RUL_{max} is a constant numeric value set for each sub-datasets differently according to the observation of their signal sensors data.

As Equation 3.3 shows, the new RUL target function limits the maximum value for the RUL which will help us to prevent the model from overestimating the RUL. It also overlaps better with signal sensor data trend which is depicted in Figure 3.4 as the degradation of the system normally happens when the system reaches a certain point.

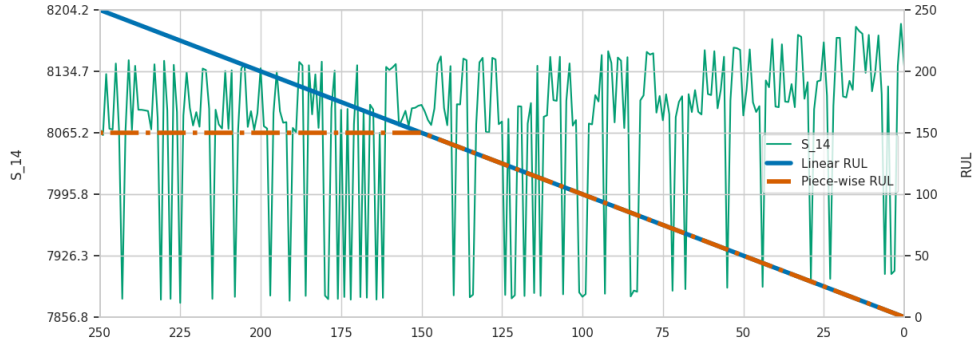
$$RUL_i(j) = \begin{cases} RUL_{max} & \text{if } t_j \geq RUL_{max} \text{ , healthy stage} \\ N_i - t_j & \text{if } t_j < RUL_{max} \text{ , degradation stage} \end{cases} \tag{3.3}$$

This piece-wise RUL target function has been used in many other studies [80, 81, 82, 83] and was shown to be more realistic than the linear version. With our dataset, the constant numerical value of the RUL_{max} is set to 130 and 150, respectively, for sub-datasets FD001 and FD004. These RUL_{max} values were determined empirically by observing [59, 54, 57] the critical point on the sensor signals trend, which at that point, the degradation starts to appear. As shown in Figure 3.4, on sub-dataset FD001 and for sensor S_7, the deterioration and change in sensor reading visibly starts after passing almost 130 cycles before failure. The same change in behavior and shift in sensor reading trend is also visible on sub-dataset FD004 and for sensor S_14 after passing the point that there are 150 more

cycles till engine failure.



(a) FD001



(b) FD004

Figure 3.4: Illustrating Linear and Piece-wise RUL Target Functions and Sensor (S_7 and S_14) Signal in the C-MAPSS Dataset

Choosing the maximum RUL for practical applications at a point where early failure signaling affects operations and maintenance schedules is recommended since doing the maintenance at an earlier stage is still less expensive than late maintenance and failure. If the maximum RUL is set too low, planning maintenance operations may be impossible and even hazardous in terms of safety. Adopting a maximum RUL that is too high leads to inaccurate projections, or it may not be required to know because maintenance procedures are already being performed at a higher interval.

3.5 Feature Selection

As stated in Section 3.4, each sub-dataset of the official C-MAPSS dataset has 3 operating conditions alongside 21 sensor signal measurements for each engine time cycle. We consider the operating conditions as three sensor signals, op_cond_1, op_cond_2, and op_cond_3 because the variation in their values affects the RUL. However, not all 21 other features significantly impact the degradation process and, accordingly, the RUL prediction.

As more and more complex and advanced models have been introduced, a larger range of sensor data can be easily deployed. Although dealing with a large number of features has become more feasible today, recent studies [84] have encouraged work on dimensionality reduction of the data by extracting the most representative and beneficial features. In addition, reducing the feature space will lead to reduced computational complexity, which in return will improve the prediction phase's performance.

One argument can be put forth regarding the overlapping role of feature selection and the use of the attention mechanism. In principle, the attention mechanism should be able to learn and differentiate between uninformative and impactful features. This characteristic of the attention mechanism is not opposing feature selection in general, as the attention mechanism helps to assign proper weights to features according to their importance. On the other hand, feature selection may eliminate features (filter methods)from the input or in some cases generate a sub-set of features (wrapper methods) that can implicitly help to increase the model's performance.

We will describe the feature selection we performed for each sub-dataset FD001 and FD004 separately. We will first discuss the steps taken on feature selection on FD001 training data and then will show the same analysis and the output for FD004.

3.5.1 FD001

For a better illustration, Figure 3.5 shows the distribution of the values of each of the 3 operating conditions values and 21 sensor readings for all the time cycles of all the engines in FD001. The y axis of each graph shows the domain of the values of the feature, and the box plot illustrates the spread and locality of the readings. This gives us an overview of how the readings from each sensor are distributed in their domains.

On the other hand, Figure 3.6 shows the variation of the feature values for all engine units for all 3 operating conditions alongside 21 sensor signal values through the engine life cycles. Each sequence from an engine is assigned a specific color to help differentiate the trends from each other. As we can see some engines have a longer life cycle and their signal readings start at earlier points. At the end of the engines' life cycle ($x = 0$), we can see some dramatic changes in the trend of some sensors. These are the sensors that contribute more to RUL prediction as their behavior and readings change according to the engine degradation stage.

For the sake of dimensionality reduction and increasing the prediction model's effectiveness, we need to select only the sensors that contribute to the RUL prediction process the most and deliver helpful degradation information. As shown in Figures 3.5 and 3.6, the signals from the seven sensors S_1, S_5, S_6, S_10, S_16, S_18, and S_19 do not show any variation through the engine cycles. In other words, there is no correlation with the time on these sensors through the engine's lifetime, and the measurements are always constant.

To accomplish the goal stated above, we considered these seven monotonic sensors as candidates for removal.

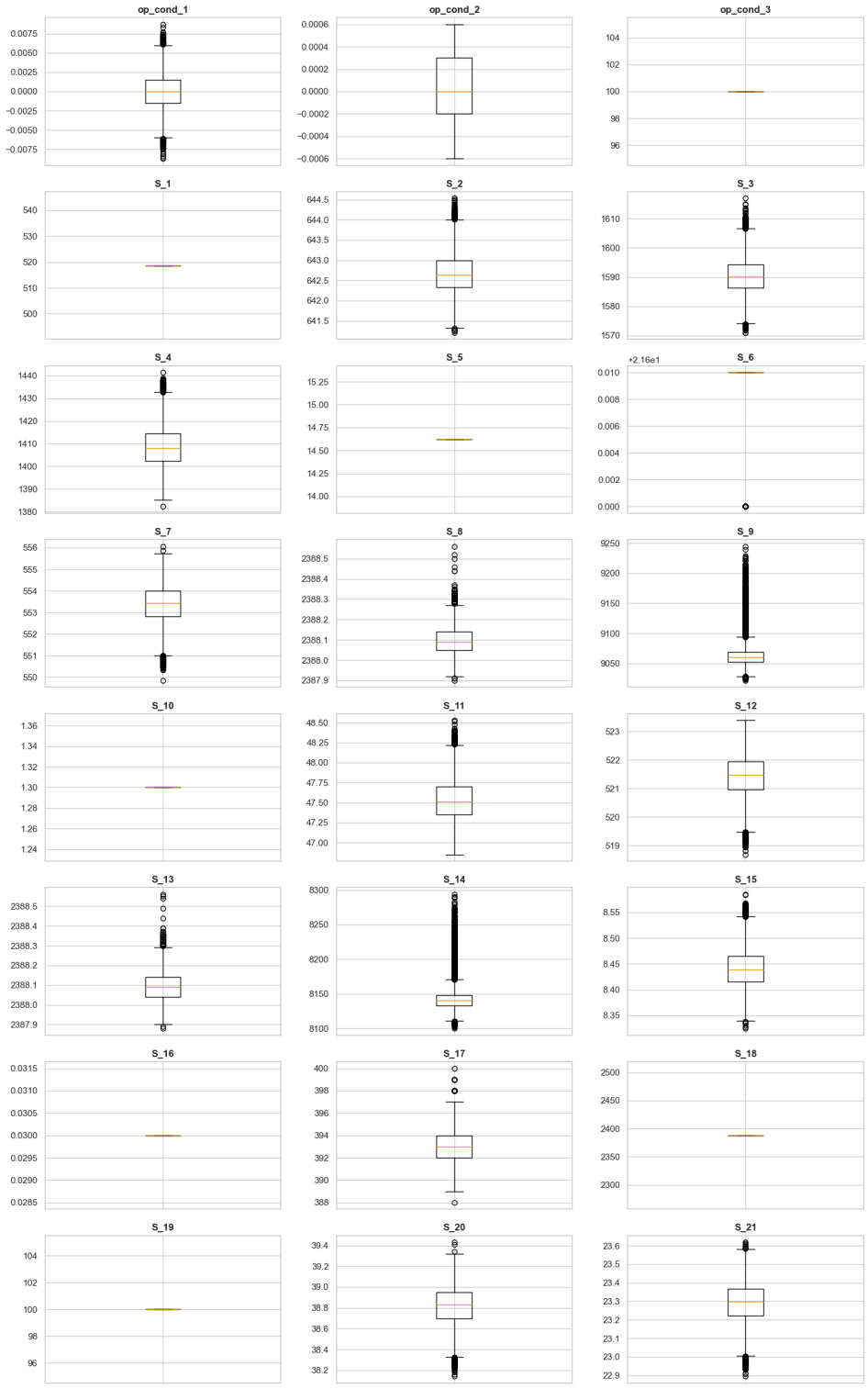


Figure 3.5: Illustration of All Engine Units Range and Distribution for Three Operating Conditions Plus Sensor Signal Sequences for FD001

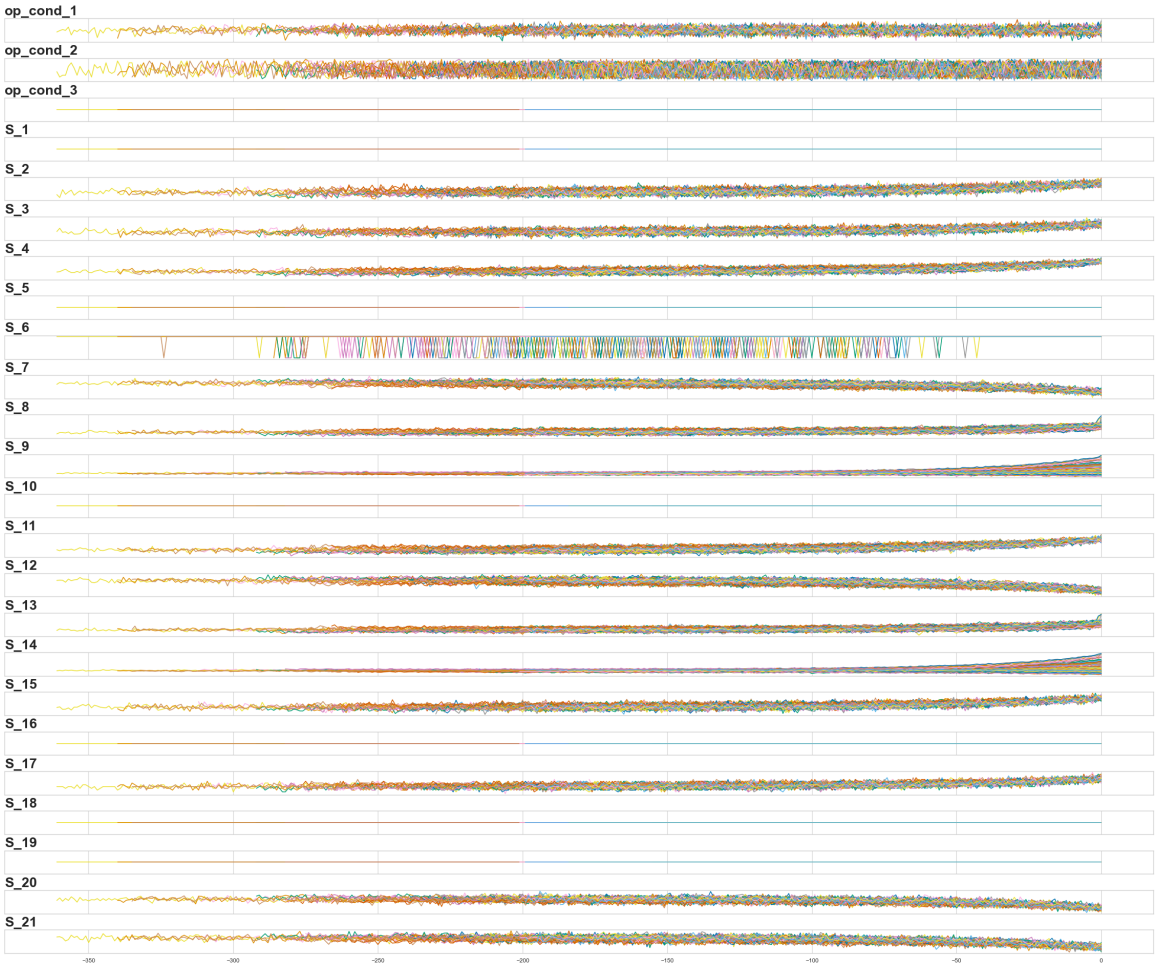


Figure 3.6: Variation of Feature Values of All Engine Units Operating Conditions Plus Sensor Signal Sequences for FD001

In order to confirm the lack of influence of these features, we computed the correlation of all the features with one another. This is shown in Figure 3.7. As the figure shows, the same features (S_1, S_5, S_6, S_10, S_16, S_18, and S_19) showed zero correlation to the other ones. So by considering the monotonic behavior and no correlation to the other features we can say their impact on the RUL estimation is also almost zero. The weak correlation shown with S_6 is due to fluctuating over a very short range of values for different engines. As we can see, for the rest of the features, there is not much more room to do on the feature selection task as the level of correlation and the distribution is not strong enough to act further.

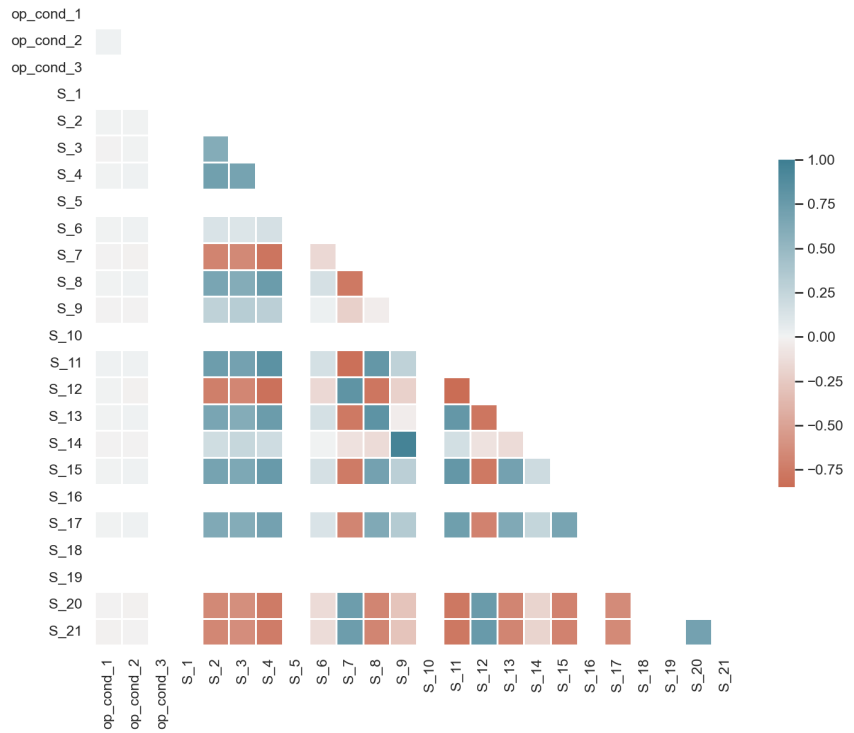


Figure 3.7: Pearson Correlation Matrix for All the Engine Units in FD001

Based on the above analysis, we removed the above-mentioned sensors (S_1, S_5, S_6, S_10, S_16, S_18, and S_19) from our sub-dataset, and the removal was effective according to what we will show in Section 4.3. The impact of removing these sensors from the input space is also studied in [79, 85, 86]. As a consequence, each time series data will have 3 operating conditions corresponding to the output of op_cond_1, op_cond_2, and op_cond_3 plus 14 features corresponding to the outputs of 14 sensor signals (21 original-7 removed). The sensors are S_2, S_3, S_4, S_7, S_8, S_9, S_11, S_12, S_13, S_14, S_15, S_17, S_20, S_21, so the total number of features on FD001 is 17.

3.5.2 FD004

Figure 3.8 shows the distribution of the values of each of the 3 operating conditions values and 21 sensor readings for all the time cycles of all the engines in FD004. The y axis of each graph shows the domain of the values of the feature, and the box plot illustrates the spread and locality of the readings. This gives us an overview of how the readings from each sensor are distributed in their domains.

On the other hand, Figure 3.9 shows the variation of the feature values for all engine units for all 3 operating conditions alongside 21 sensor signal values through the engine life cycles. Each sequence from an engine is assigned a specific color to help differentiate the trends from each other. As we can see, some engines have a longer life cycle, and their signal readings start at earlier points. At the end of the engines' life cycle ($x = 0$), we can see much less dramatic changes in the trend of some sensors in comparison to FD001. As Figure 3.9 further shows, the level of involvement from sensors in FD004 is much higher than in FD001, so there is no clear trace of dependency on most of the FD004 sensor readings. This dramatic change in the behavior and the trends of the sensor readings is due to the impact of six conditions and two fault modes in the FD004 sub-dataset.

As stated before, to reduce dimensionality and to increase the prediction model's effectiveness, we need to select only the sensors that contribute to the RUL prediction process the most and deliver helpful degradation information. As shown in Figures 3.8 and 3.9, the signals from the two sensors S_13 and S_19 do not show much variation through the engine cycles. In other words, there is no correlation with the time on these sensors through the engine's lifetime, and the measurements are always constant. The change in the trend line for both of the sensors is due to fluctuating over two values on different engines.

To accomplish the goal stated above, we considered these two monotonic sensors as candidates for removal.

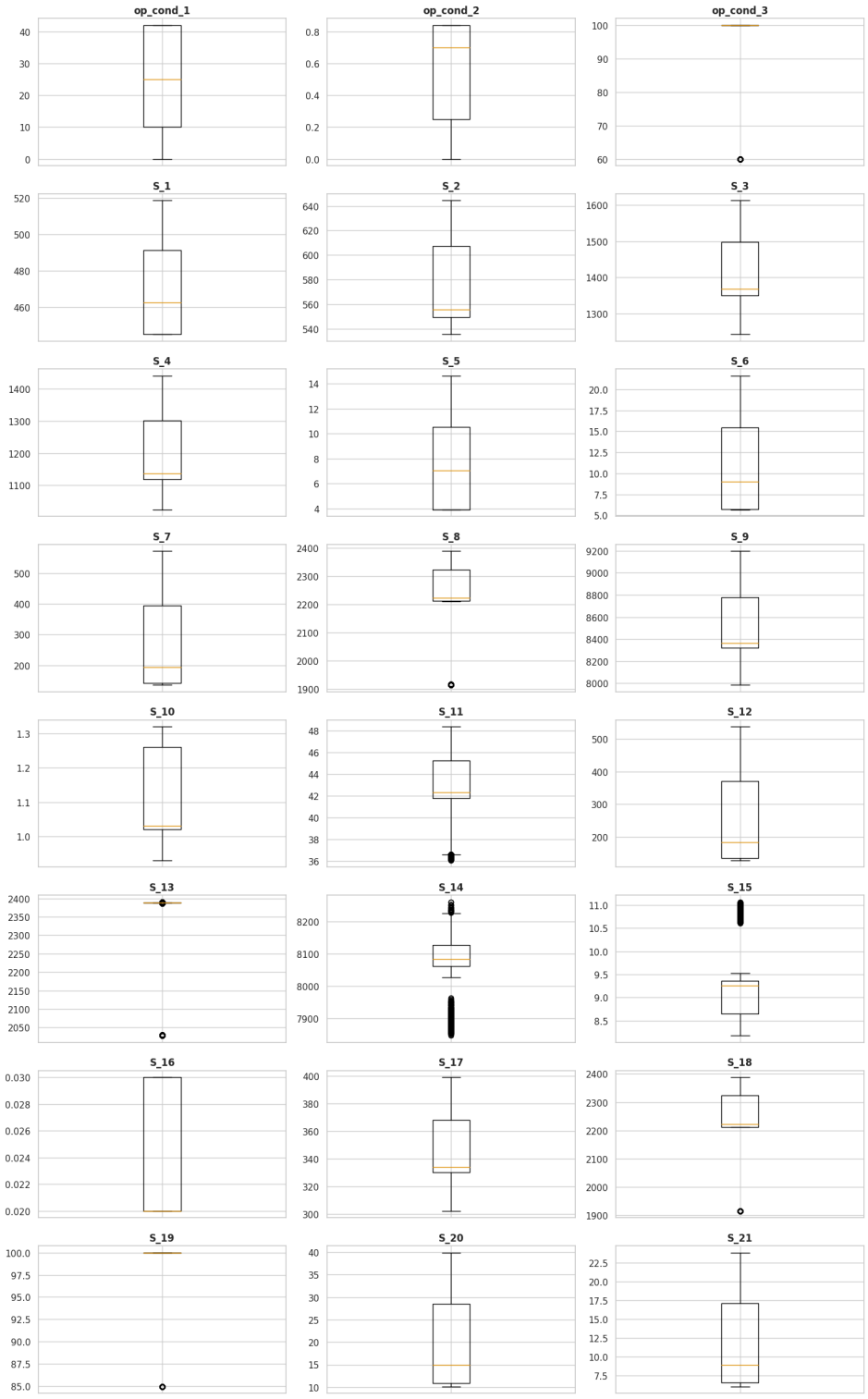


Figure 3.8: Illustration of All Engine Units Range and Distribution for Three Operating Conditions Plus Sensor Signal Sequences for FD004

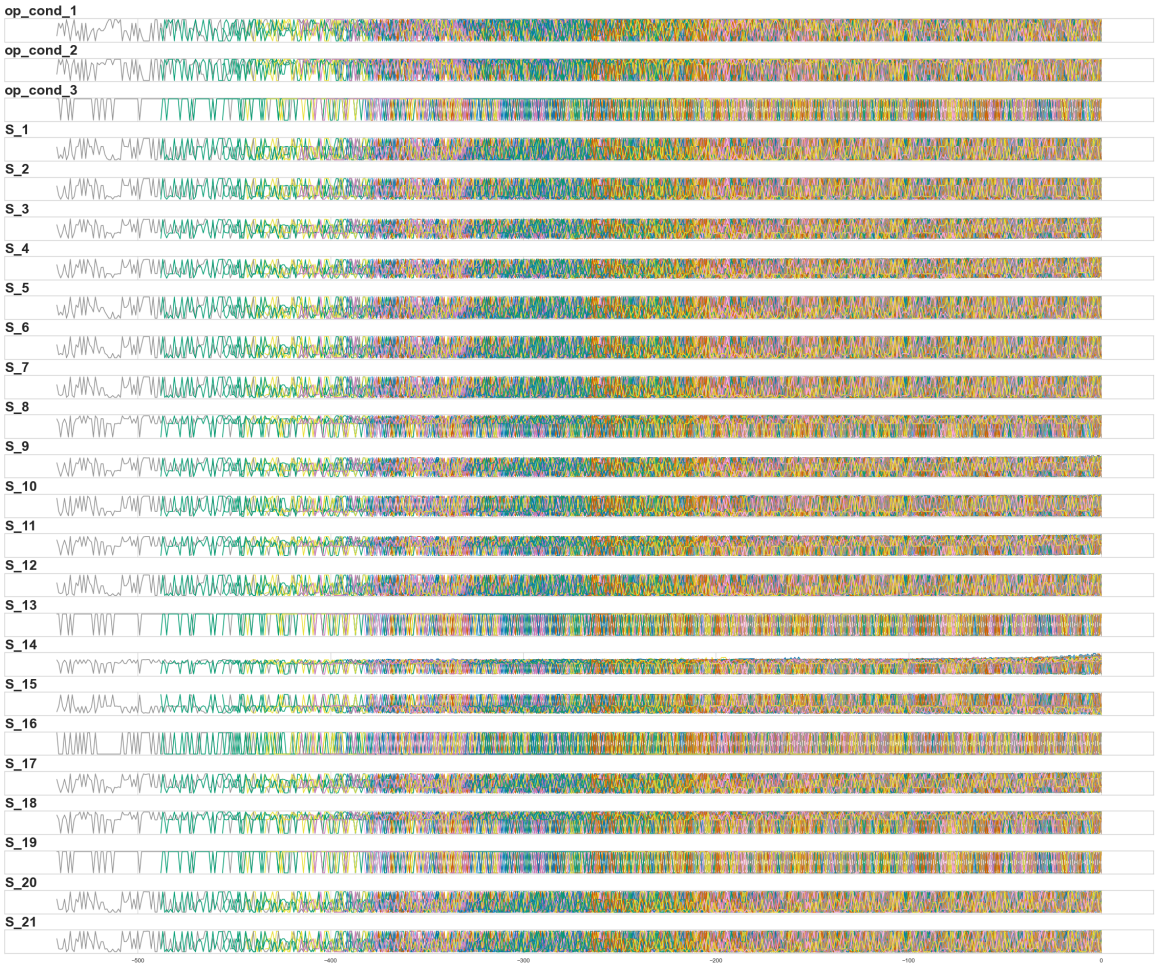


Figure 3.9: Variation of Feature Values of All Engine Units Operating Conditions Plus Sensor Signal Sequences for FD004

In order to further investigate the lack of influence of these features, we computed the Pearson correlation of all the features with one another. This is shown in Figure 3.10. As the figure shows, the features S_13 and S_19, unlike what we saw in FD001, have quite a strong correlation to other sensors and operating conditions. Due to such interconnectivity, we did not remove these two sensors from the list of features. For the rest of the features as well, we can clearly observe that the level of correlation between sensors is quite high.

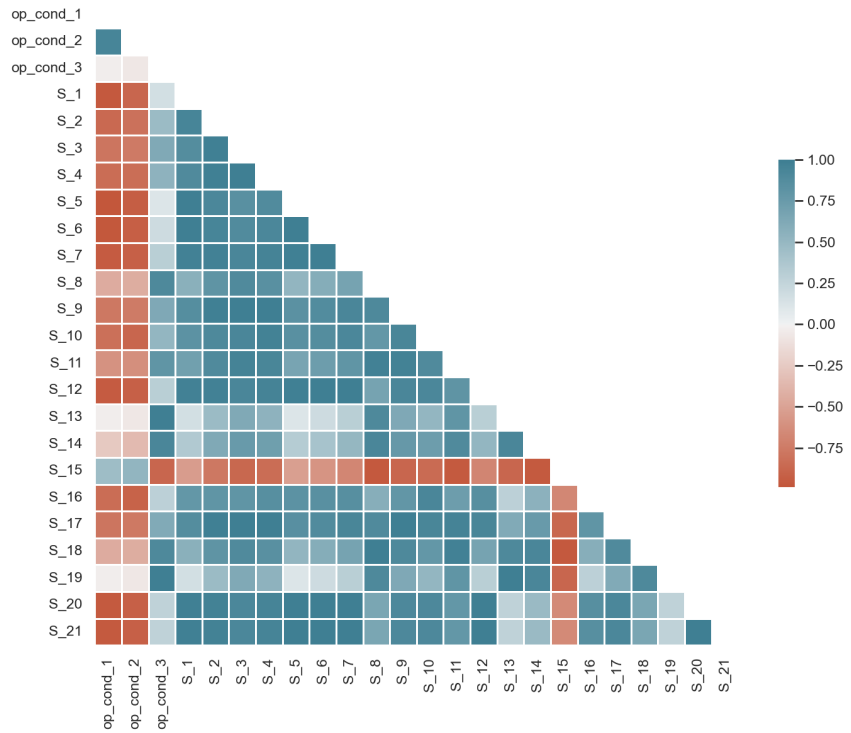


Figure 3.10: Pearson Correlation Matrix for All the Engine Units in FD004

Based on the above analysis, we kept all the sensors in the features list, and therefore for FD004 we have a total of 24 features: 3 operating conditions alongside 21 sensors.

3.6 Normalization

To feed the input features to our proposed model, we need to normalize the input data to reduce the bias effect that the contribution of different scales of variables may cause. The raw sensor signal values fall into different numerical ranges, which will also cause large weight updates on backpropagation and a longer duration of the training. The raw sensor signals on our sub-datasets also have different domains that need to be normalized. The difference in sensor signal domains after normalization is corrected and all sensor readings

are scaled.

Usually, normalizing the input data will prevent the exploding gradient problem [48] that can be caused by raw input data. The exploding gradient is a well-known issue in the training phase of gradient-based neural networks on the backpropagation step. This happens when gradients keep getting larger through backpropagation step progression. This problem, as stated above, will cause large weight updates and divergence of the gradient descent. We considered both Z-score normalization, also known as standardization [83], and min-max normalization [87], scaling methods to eliminate the presence of the bias in the input data and also to improve the convergence speed of the model. As discussed in Section 3.5 and depicted in Figures 3.5 and 3.6, the outliers are very rare in the C-MAPSS dataset (almost no outliers). This will give us a better result by normalization of the raw inputs. As having outliers causes the normalization result to be affected by bias from the outliers' values. One important advantage of min-max normalization rather than Z-score normalization is that it guarantees that all input data will fall into the same range ([0, 1]). Equation 3.4 shows Z-score normalization.

$$x'_i = \frac{x_i - \mu_i}{\sigma_i + \epsilon} \tag{3.4}$$

where x'_i is the normalized feature value, x_i is the original feature value for the i^{th} sensor measurements, μ_i is the mean value of the i^{th} feature, and σ_i is the feature standard deviation for the time series data. As we are using a piece-wise RUL target function and to avoid sample imbalance, we need to consider that a large number of training samples will have the same value of RUL_{max}. The ϵ is added in the denominator of Equation 3.4 first to set the maximum value of the normalized data slightly less than the true maximum value and also prevent the denominator to become 0 (division by zero) [88].

Equation 3.5 shows min-max normalization.

$$x_i' = \frac{x_i - \min x_i}{\max x_i - \min x_i + \epsilon} \tag{3.5}$$

where $\max x_i$ and $\min x_i$ are the maximum and minimum values of the i^{th} feature for the time series data. The ϵ is added for the same reason as in Equation 3.4.

3.7 Time Window Processing

In order to make use of historical data, we used a fixed-size time window to encapsulate multi-data points sampled continuously. A time window is commonly used for data segmentation. The data retrieved from the CMAPSS dataset is a long list of values. This dataset must be split to include each engine separately. In Section 4.4.4 we will show the result of applying different sliding time windows on the dataset. The next step is to apply the time windows. An example of data segmentation for a time window is shown in Figure 3.11. This frame is equal to the number of selected sensors (n) by the size of the window (s). This window is then placed p time steps further to obtain the next data frame. This results in a number of data points equal to the length of the life span of the engine (T) minus the window length (s) of data samples per engine (i.e., $T - s$). So the RUL of the $(i + 1)^{th}$ sample is shown in Equation 3.6.

$$RUL_{i+1} = T - s - (i \times p) \tag{3.6}$$

Many multivariate temporal data-based prediction models take a multivariate data point sampled at a single time stamp as an input [54]. This strategy neglects useful temporal information that may improve prediction performance. To address such an issue, we utilized a fixed-size time window (TW) to enclose multivariate data points sampled at consecutive timestamps. Specifically, at any timestamp, multivariate data points within the TW that

covers the current timestamp and its preceding timestamps are concatenated into a high-dimensional feature vector which is then fed as input into the prediction model. In practice, a suitable TW size can be chosen via cross-validation. However, for this study, we have chosen 30 as our window size ($s = 30$) and considered our step size to be 1 ($p = 1$), which are the same as the parameters used in [59]. With these settings, we change the number of training samples for dataset FD001 from 20631 to 17731 and from 61249 to 57522 for dataset FD004. Obviously, for the testing step, we only consider a one-time window for each engine, giving us the same number of testing samples as the number of engines in the corresponding datasets.

By going through normalization and applying a sliding time window to our samples, our training data is finally prepared for feeding to our proposed model.

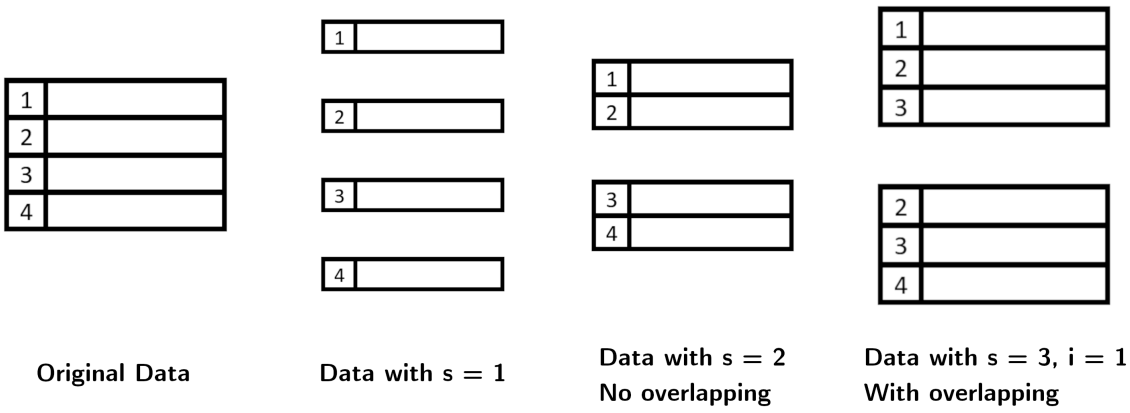


Figure 3.11: Illustration of the Sliding Time Window Used to Encapsulate Multi-data Points

3.8 Evaluation

We require some objective performance measurements to fairly compare the estimation model performance on the dataset. Due to the characteristics of prognostic problems, we need to evaluate early and late predictions differently. Late predictions need to be more

penalized because late maintenance can cause severe problems and critical failures rather than early predictions, which would lead to a waste of resources. Hence, we need to take into account this requirement when dealing with the evaluation of a model and add more weight to penalizing late predictions.

Considering the points discussed above, we used two widely used metrics whose effectiveness has been shown by several studies, such as [79, 89]. These two functions were introduced to fairly measure and evaluate the performance of the models proposed for the C-MAPSS dataset. The first metric we considered for evaluation is a scoring function first proposed by [90], which considers early and late predictions and penalizes more heavily late predictions to measure the model performance as stated above. Equation 3.7a shows the definition of this asymmetric function. Where S is the computed score, N is the total number of samples in the dataset, and d_i is defined in Equation 3.8. The constant values are given by the dataset creator [90] to differentiate the penalty on late and early predictions.

$$\text{Score} = \sum_{i=1}^{N} S_i \tag{3.7a}$$

$$S_i = \begin{cases} e^{-\frac{d_i}{13}} - 1 & \text{for} \quad d_i < 0 \\ e^{\frac{d_i}{10}} - 1 & \text{for} \quad d_i \geq 0 \end{cases}, i = 1...N \tag{3.7b}$$

$$d_i = \overline{RUL_i} - RUL_i \tag{3.8}$$

Nevertheless, this evaluation has a few drawbacks. The main one is that a single outlier on a very late prediction would significantly affect the value of the score and the overall performance level. This will be caused by exponential growth in the scoring function, as illustrated in Figure 3.12 by the orange curve. Another drawback of this scoring function is the lack of consideration of the prognostic horizon of the algorithm. The prognostic

horizon assesses the time before failure and the algorithm can accurately estimate the RUL value within a certain confidence level [54].

The other metric is the well-known root-mean-square error (RMSE) that is defined in Equation 3.9. Where same as the scoring function, N is the total number of samples in the dataset, and d_i is calculated according to Equation 3.8. RMSE is widely used for performance evaluation in regression models. The RMSE function gives equal weight to early and late predictions, which serves the above-mentioned point. Using RMSE alongside the scoring function of [90] eliminates the tendency to favor an algorithm that artificially lowers the score by underestimating it but resulting in higher RMSE. Graphically, RMSE is shown in Figure 3.12 by the blue curve.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} d_i^2} \tag{3.9}$$

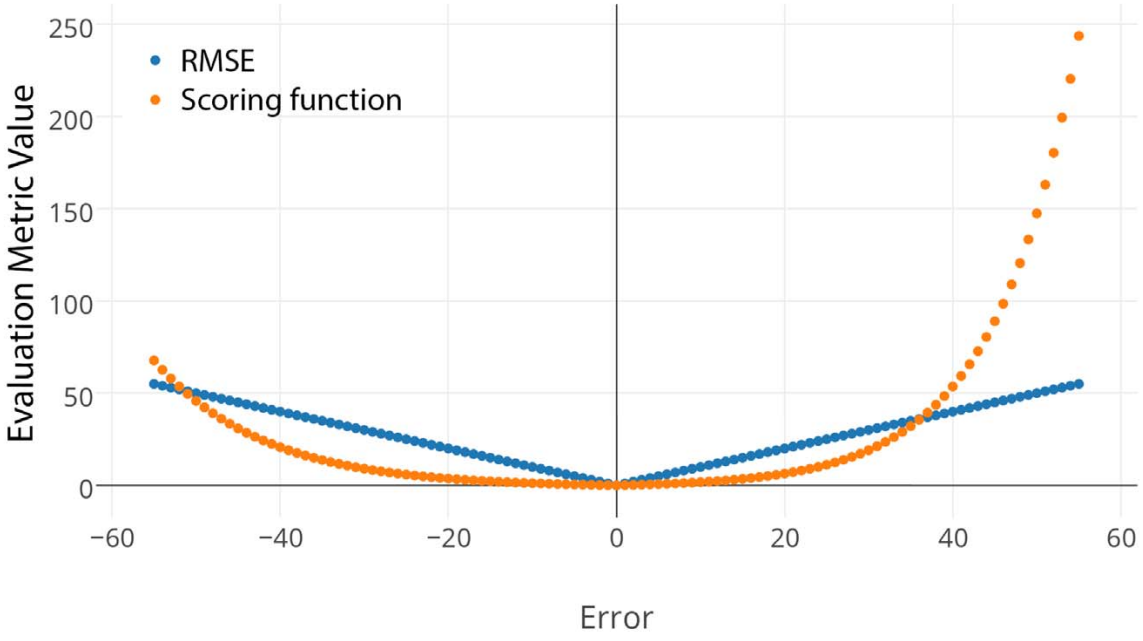


Figure 3.12: Illustration of the Scoring Function of [90] and RMSE

3.9 Chapter Summary

In this chapter, we first introduced the C-MAPSS dataset that simulates turbofan engines. We described how these datasets are shaped and the corresponding settings for each one. After that, we briefly introduced the notations used in our thesis and time series data. In Section 3.3, we discussed the operating conditions and sensors in the C-MAPSS datasets in detail. We described all four C-MAPSS datasets and their corresponding operating conditions and how these datasets differ from each other. Finally, we reviewed the list of all available sensors for turbofan engines.

In Section 3.4, we defined the target function of the problem and the label of datasets as the remaining useful life (RUL) of an engine. We discussed different approaches for formulating the RUL and defined a piece-wise target function. Then, we described feature selection. We first showed the raw sensor data and how they are distributed. Then, we identified monotonic sensors and removed them from the target set of selected features. After that, we analyzed the possible correlations between pairs of features. We finally normalized the features to reduce the bias effect from different sensor data domains and to scale our raw features. We deployed min-max and Z-score normalization methods to normalize the sensor signal readings and to eliminate the impact of bias from sensor readings having different scales.

In Section 3.7, we introduced the time window processing technique and applied it to our targeted datasets. By having our features normalized and adding the time window, the training data was prepared to be fed to our proposed neural network model. Lastly, in Section 3.8, we discussed the metrics used for evaluating our model. We introduced RMSE and the scoring function of [90], which penalizes late predictions more than early predictions due to the nature of prognostic problems.

In the next chapter, we will describe the architecture of our proposed model, and present

all experiments and fine-tuning procedures that we conducted according to our setup. Finally, we will evaluate the performance of the proposed model and compare it with other models in the same predictive maintenance domain.

Chapter 4

Experiments

This chapter discusses our experiments. We first describe our methodology, architecture setup, and implementation details, then report on the evaluation of the proposed model. This chapter is structured as follows: In Section 4.1, we discuss the architecture of our proposed method. In Section 4.2, we present the implementation details of the proposed model and the hyperparameter settings. In Section 4.3, the results of the experiment on different datasets are reported. In Section 4.4, we show the effect of different hyperparameters on the experiment result. Finally, in Sections 4.5 and 4.6 we compare the proposed model to other recent models and discuss different aspects of the experiments.

4.1 Proposed Models

Remaining useful life (RUL) prediction based on the attention mechanism is a relatively new approach that has shown promising results in recent studies [65]. The basic idea behind this approach is to use an attention mechanism to identify the most relevant features or time steps in the input data that are predictive of the RUL and to weigh these features or time steps accordingly.

One common way to apply attention mechanism to RUL prediction is to use a recurrent

neural network (RNN) or a similar sequential model, where the input data consists of time-varying signals such as sensor readings or other measurements. The RNN processes the input data sequentially. At each time step, it uses an attention mechanism to compute a set of attention weights that reflect the importance of each feature or time step for predicting the RUL.

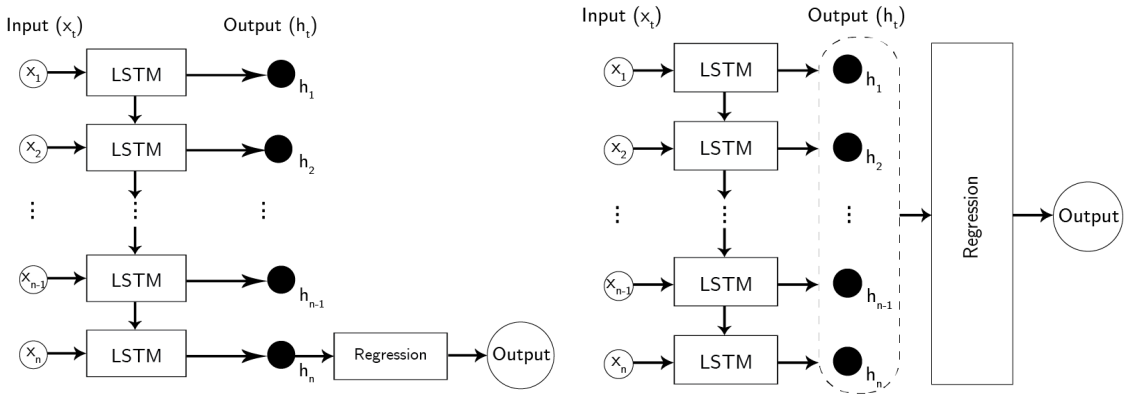
The attention weights can be computed using various techniques, such as linear or non-linear activation function (neural network), a dot product, or other methods (see Section 2.1.6). Once the attention weights are computed, they are used to weigh the input data, which is fed to a linear layer to predict the RUL.

One advantage of this approach is that it can identify the most important features or time steps in the input data, which can be useful for interpreting the results and understanding the underlying factors contributing to the RUL. It can also improve the performance of the RUL predictions by allowing the model to focus on the most relevant parts of the input data and filter out noise or irrelevant features.

However, attention-based models can also be more complex and computationally intensive compared to other approaches and may require more data and longer training times to achieve good performance. Nonetheless, with the increasing availability of large datasets and powerful computing resources, attention-based RUL prediction models are becoming increasingly popular [91] and a promising approach in the field of predictive maintenance.

Due to the inherent sequential modeling ability of the LSTM network, it has been successfully used for machine RUL prediction in [57]. However, conventional LSTM only uses the learned features at the last time step for regression. This is shown in Figure 4.1 (a). By considering the concept of attention, we propose that the learned features from earlier time steps could potentially be useful and the network hidden state on each time step can contribute to the final output of the network, which is the modification depicted in Figure 4.1 (b). This comes from the idea that on any time series data we may have different

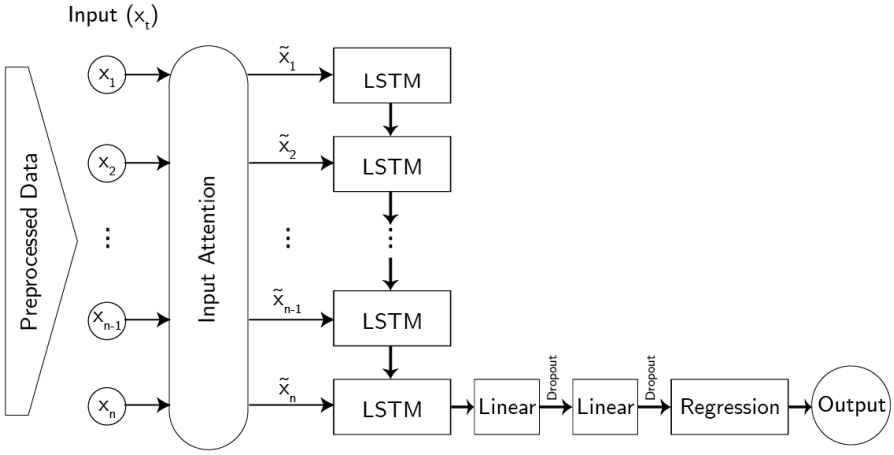
periods that consist of more important information and also, on the other hand, some sequences include more redundant features. Additionally, the learned traits could contribute differently to the final RUL prediction. So that we may understand the significance of characteristics and time steps, we used an attention mechanism.



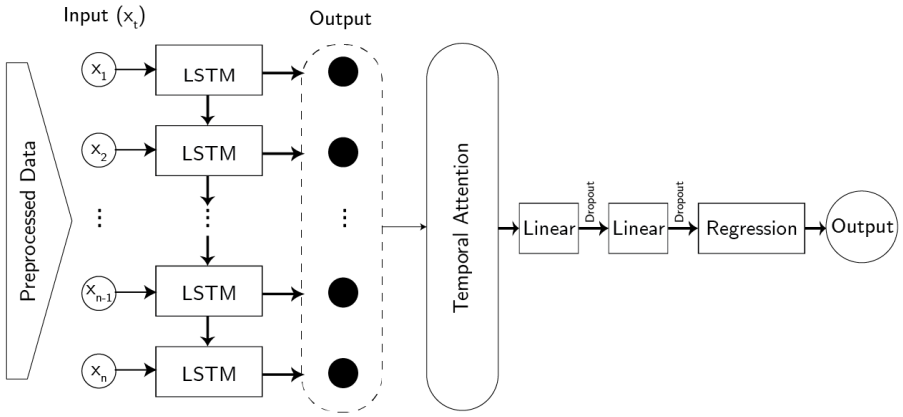
(a) Standard LSTM Architecture Setting (b) Proposed LSTM Architecture Setting

Figure 4.1: LSTM Architectures Used for RUL Prediction

We implemented both a standard LSTM and our proposed LSTM with a self-attention block to evaluate the effectiveness of the proposed modifications. We conducted the first experiment (model A) by applying the self-attention layer only to the input data. The architecture for model A is shown in Figure 4.2a. One advantage of applying the attention layer to the input was to understand the domain clearly. For the second experiment (model B), we only applied the self-attention layer to the output of the LSTM layer. The architecture for model B is shown in Figure 4.2b. The application of the attention layer only on the output of the LSTM layer helps to consider the importance of the temporal data.



(a) Model A



(b) Model B

Figure 4.2: Illustration of the Proposed Architectures for Models A and B

Finally, in the third experiment (model C), we applied the self-attention layer to both the input layer data and the output of the LSTM layer. As shown in Figure 4.3, the LSTM network is first loaded with the output of the attention layer on preprocessed sensory input data to learn a feature representation. The attention layer's outputs or attention weights, which embed the significance of each feature and time step, are calculated based on the learned sequential features as input. The attention block's outputs are combined with the learned sequential features obtained from the LSTM layer. After that, we feed the output into two consecutive fully connected layers (FC), also known as linear layers. This helps for more abstraction of features and also as we are dealing with a non-linear problem,

we deployed multilayer perceptron linear layers (2) to represent the convex regions for overcoming the linear separability. This helps change the dimensionality of the output from the preceding layer so that the model can easily define the relationship between the values of the data in which the model is working. Finally, we use a regression layer to predict the RUL.

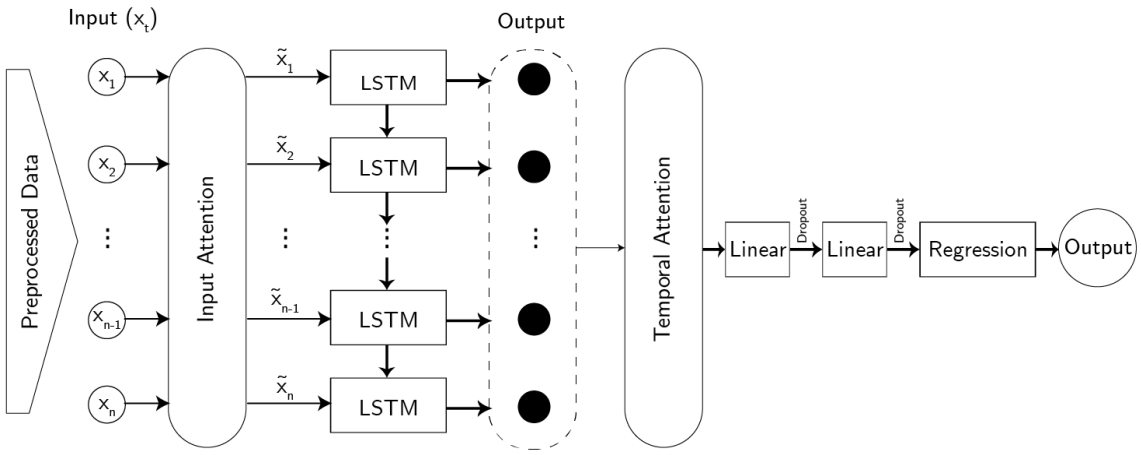


Figure 4.3: Architecture of the Proposed Attention-based Deep Neural Network (Model C)

The results with the FD001 and the FD004 datasets are reported in Table 4.1. As we can see with both the RMSE and the Score metrics, the modifications significantly enhance the network's performance. The impact of both attention layers, especially the temporal one, was more significant on FD001 than on FD004. This may indicate that more complex datasets may predict a more accurate RUL by having more information. Based on the result of the experiments we can clearly see that the self-attention layer helped in all proposed models and the best performance belongs to the model C as we have both input layer and temporal layer attention.

Table 4.1: Result of the First Experiments

Model	Architecture	FD001		FD004	
		RMSE	**Score**	**RMSE**	**Score**
Baseline	Standard LSTM + 2 Linear Layers + Regression	15.43	410.60	29.12	12551.44
A	Input Attention Layer + Standard LSTM + 2 Linear Layers + Regression	14.11	331.60	28.84	11041.63
B	Proposed LSTM + Temporal Attention Layer + 2 Linear Layers + Regression	13.96	297.38	27.33	7354.39
C	Input Attention Layer + Proposed LSTM + Temporal Attention Layer + 2 Linear Layers + Regression	**13.27**	**270.84**	**25.16**	**5065.06**

4.2 Hyperparameter Settings

All four methods in Table 4.1 used the same hyperparameters. To set the dimension of the LSTM and linear layers, plus considering the randomness in parameter initialization in them, we used repeated k-fold cross-validation with $k = 10$. According to the technique, in the end, we average our results that are achieved at the end of each iteration. In Table 4.2, we indicate the hyperparameters we selected for our networks and the range of each one we experimented on. In Section 4.4, we will discuss how these parameters are selected and tuned.

The next sections will analyze the experimental results. Then, the hyperparameter selection and tuning will be discussed.

Table 4.2: Hyperparameters Used for the Results Shown in Table 4.1

Hyperparameter	**Range**
Epoch	10, **30**, 100, 200
Learning rate	1e-4, 5e-4, 2.5e-4, **1e-3**, 1e-2
Optimizer	**Adam**, RMS-prop
Normalization method	**min-max**, Z-score
Time Window Size	10, 20, **30**, 40, 50, 60
Batch size	16, 32, **64**, 128
Dropout	**0.2**
Loss function	**MSE**

Recall from Section 3.8, that we used two metrics to evaluate our approach, RMSE, and the scoring function of [90]. For test purposes, we can obtain RMSE easily from the model using the mean square error (MSE). For each layer of the network, we can calculate and backpropagate the MSE loss to obtain error gradients for the particular layer. Then, based on the error gradients at each layer, the Adam optimizer is used to improve the model parameters. This method computes adaptive learning rates for each parameter. We will discuss the optimizer selection in detail, in Section 4.4.

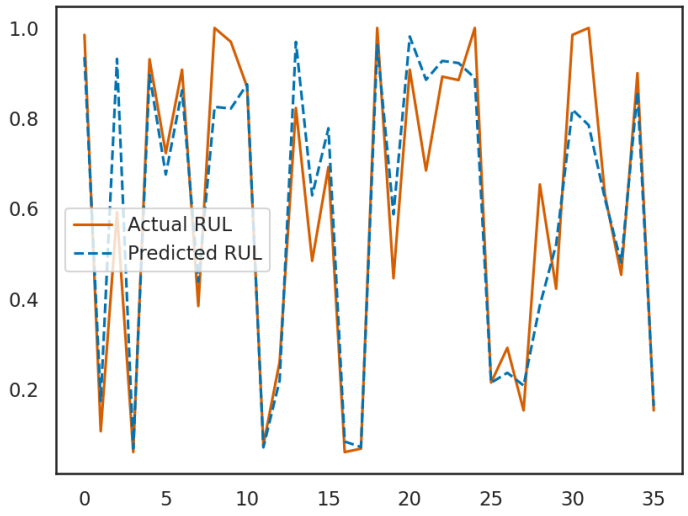
For the batch size, according to recent studies [92], we targeted a range of power of 2s candidates. For tasks where the generalization capability is essential, including image classification, natural language processing, and time-series analysis, smaller batch sizes are recommended. Larger batch sizes can be utilized for applications when training time is more of a concern. Mini-batch gradient descent, where the training data is divided into mini-batches of size n, where n is the batch size, is frequently used in practice. After each mini-batch, the model parameters are changed, and the procedure is repeated until the full training set has been processed. For this research, we applied the optimal batch

size of 64 based on the performance of the candidates. We also set 0.2 as the dropout rate as we observed higher probabilities resulted in under-learning by the network and lower probabilities showed weak effectiveness on regularization.

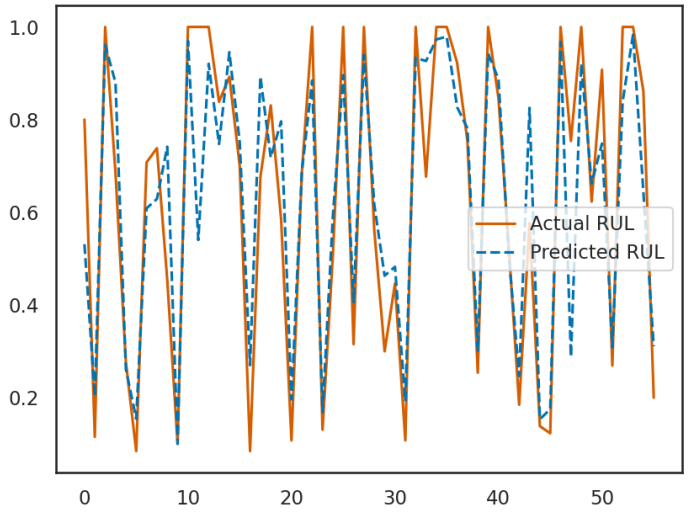
4.3 Experimental Results

The following section summarizes the proposed network's prediction performance using the hyperparameters in Table 4.2. In Figure 4.4, the predicted RULs of the testing engine units are shown (in blue) alongside the actual RUL (in orange) from test sub-datasets for FD001 and FD004. Ideally, the predicted RUL should overlap completely with the actual RUL to have the best performance. In this case, the RMSE value is minimized, but this will not guarantee model performance on its own. As we discussed in Section 3.8, in the RUL prediction problem we need to differentiate between early and late predictions. Here we can see the role of the scoring function of [90] to drive the predictions toward early predictions rather than late predictions by penalizing more the model on late ones. So the model can sacrifice an ideal accuracy and performance on RMSE and in return get better performance on scoring function relatively by not having too late predicted RULs that may cause serious issues.

As we can see from the predicted RUL of different engines, and also by considering the results reported in Table 4.1, the number of late predicted RULs is significantly lower than the number of early predicted ones (7 late predictions in FD001 and 11 late predictions in FD004 in total). This is due to the role of the scoring function of [90], introduced in Section 3.8 and the effect of the attention layer in giving more weight to the timesteps which have a higher level of importance in RUL prediction. In proper settings for a system, we should set the RUL threshold to values long before severe degradation starts to happen. This may be because sensor data deviations in the early stages of failure are harder to identify than those in the later stages.



(a) FD001



(b) FD004

Figure 4.4: Illustration of the Results of the Experiments Showing Predicted RUL alongside Actual RUL of Test Engine Units on C-MAPSS Dataset

As shown in Figure 4.4, due to the more complex and higher level of interdependency in FD004, the model output shows a better result on FD001 in comparison to FD004. As we discussed in Sections 3.3 and 3.5, the availability of two different fault modes, plus

six operating conditions alongside the correlation among the sensor data in FD004, make the task prediction much more complex and computationally intensive in comparison to FD001 data. Considering these facts, we can see how the attention mechanism contributes to the evaluation of each feature and time step, not only on FD001 with its less complex settings but also on FD004 where feature selection and temporal data analysis is quite an impractical step in the data preparation and preprocessing stage.

4.4 Effect of Hyperparameters

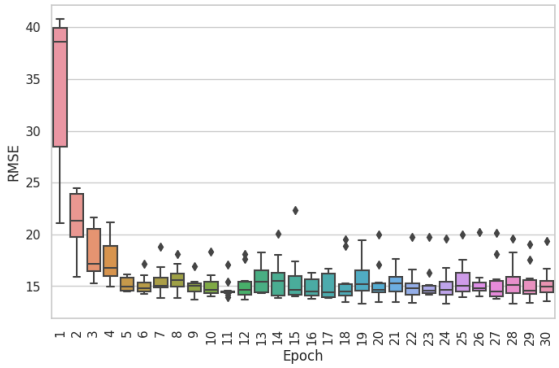
4.4.1 Learning Rate

The learning rate (LR) hyperparameter regulates how much we modify the weights of our network in relation to the loss gradient. A dynamic updating strategy for the learning rate is used in the RMS-prop and the Adam optimizer approaches. The optimizer updates the learning rate in accordance with the rate of loss determined in each update. So basically, the learning rate we discuss here is the initial learning rate.

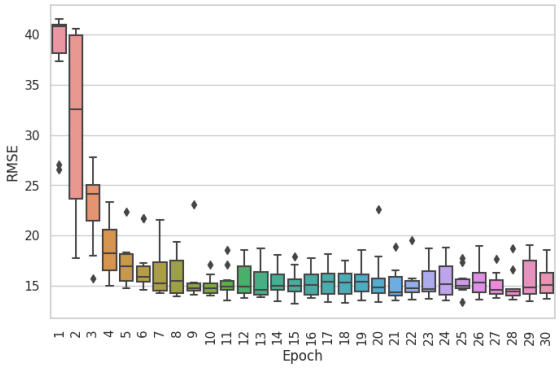
Generally, the model reaches the optimal performance region in fewer epochs by applying a too-high learning rate. The network will fail to find the optimal point, and divergent behavior will be seen in the loss function as of the drastic updates. On the other hand, the network will take too long to find the optimal point and progress when a too-low learning rate is applied, because it needs numerous updates before reaching the optimal point. That is why we need to investigate the proper learning rate for the network. A learning rate that is not too high that skips the optimal point and is not too small that takes a very long time to train and reach the optimal point.

Figure 4.5 shows the performance of the learning rates listed in Table 4.2 on dataset FD001 based on 10 experiments. As we can see, the network reached the optimal point in around 30 epochs. This is depicted in Figures 4.5a and 4.6a for the Adam and RMS-prop

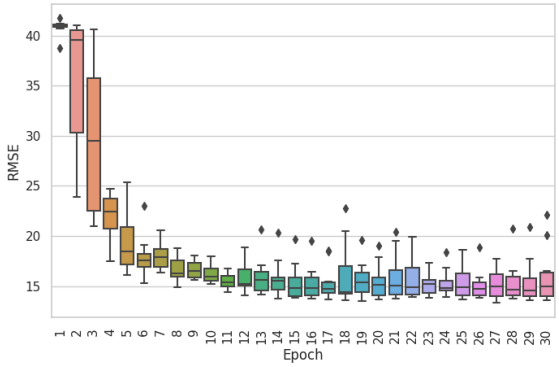
optimizers. Different colors are used to distinguish each epoch's result visibly. We compare the performance based on RMSE and the score function of [90] described in Section 3.8.



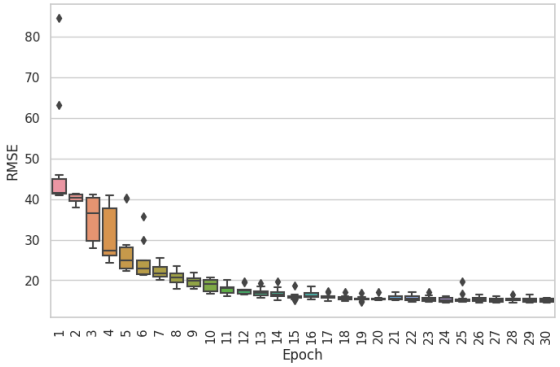
(a) LR = $1e-3$, Optimizer = Adam



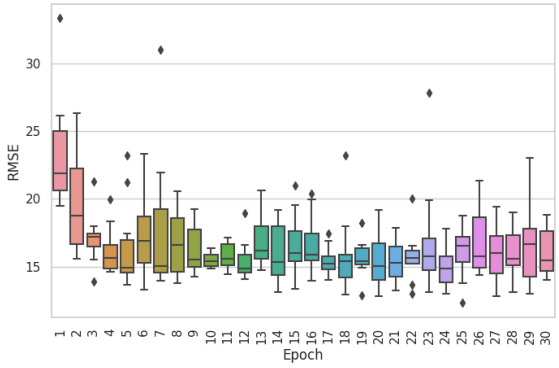
(b) LR = $5e-4$, Optimizer = Adam



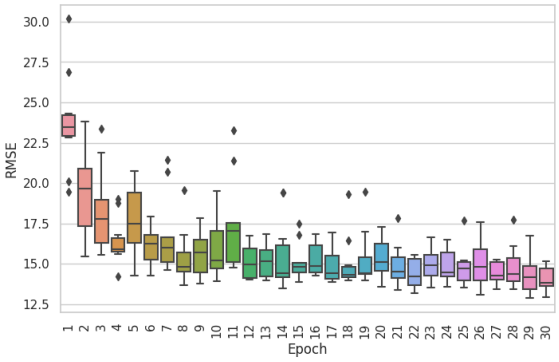
(c) LR = $2.5e-4$, Optimizer = Adam



(d) LR = $1e-4$, Optimizer = Adam

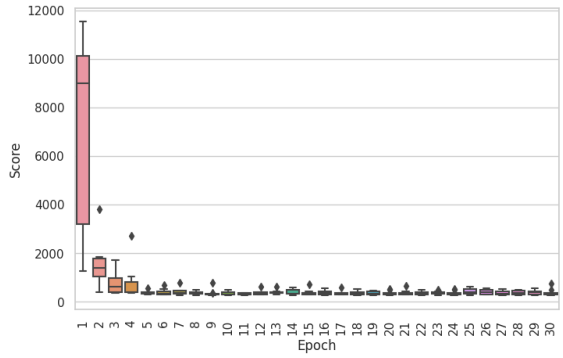


(e) LR = $1e-2$, Optimizer = RMS-prop

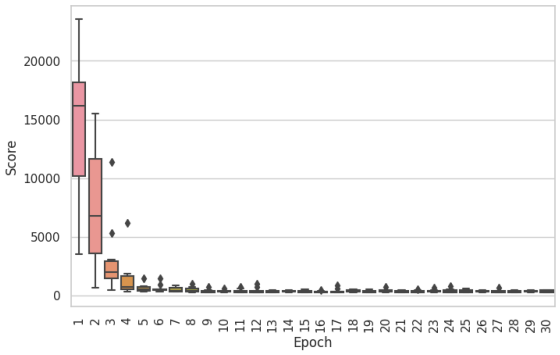


(f) LR = $1e-3$, Optimizer = RMS-prop

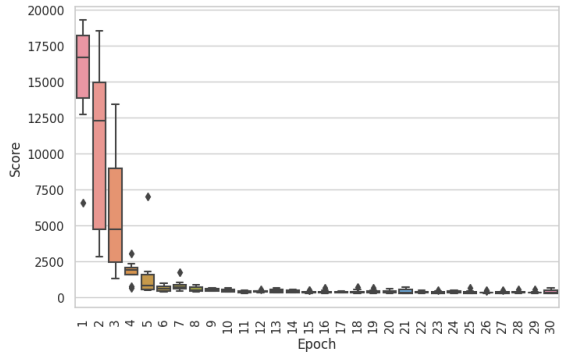
Figure 4.5: Illustration of RMSE over Epochs by Applying Various Learning Rates on dataset FD001, Based on 10 Experiments



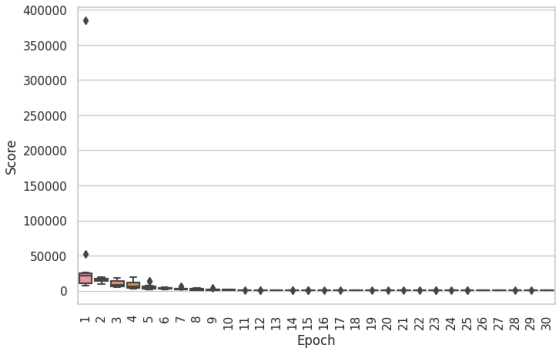
(a) LR = $1e-3$, Optimizer = Adam



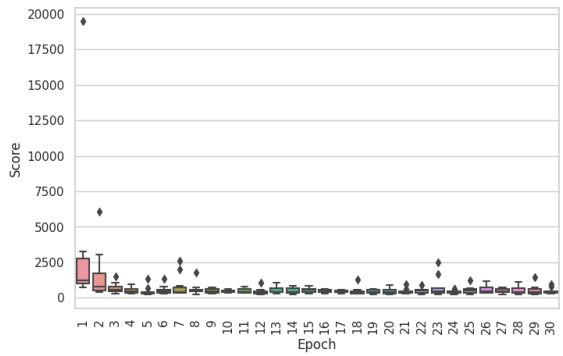
(b) LR = $5e-4$, Optimizer = Adam



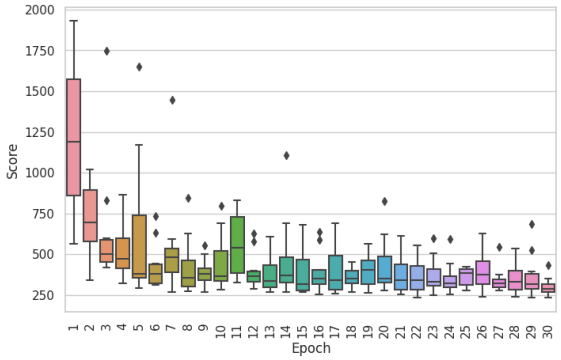
(c) LR = $2.5e-4$, Optimizer = Adam



(d) LR = $1e-4$, Optimizer = Adam



(e) LR = $1e-2$, Optimizer = RMS-prop



(f) LR = $1e-3$, Optimizer = RMS-prop

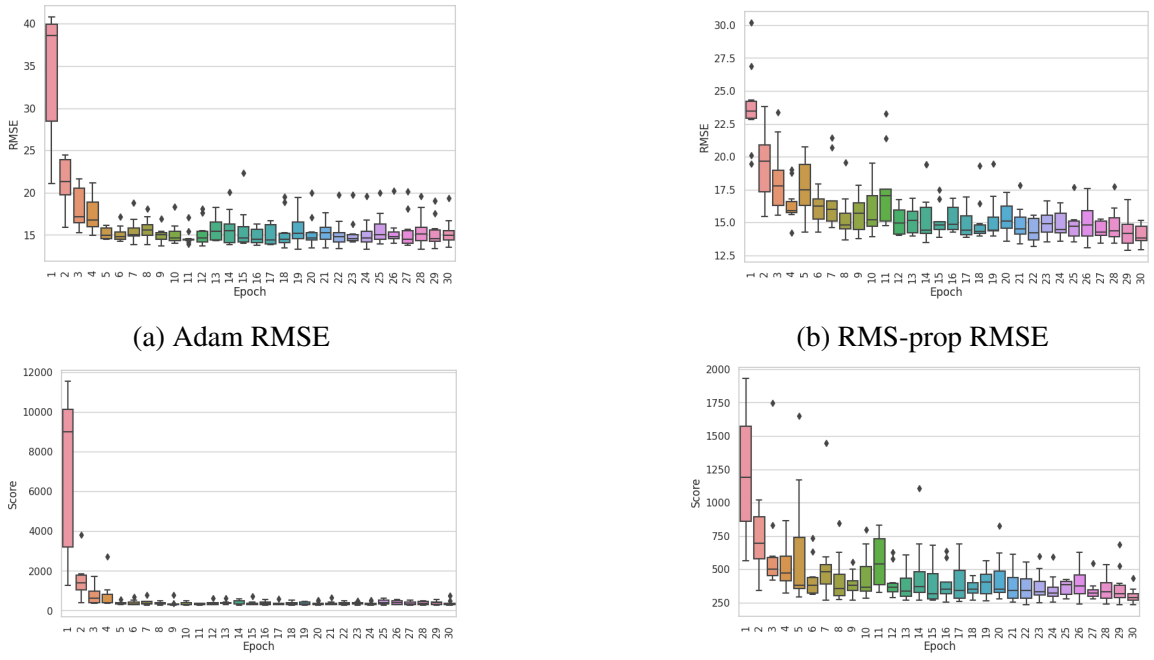
Figure 4.6: Illustration of Score Metric over Epochs by Applying Various Learning Rates on dataset FD001, Based on 10 Experiments

4.4.2 Optimizer

The optimizer we choose is one factor that might determine whether our gradient descent algorithm diverges or converges. Thanks to the development of newer optimizers [93] and

the introduction of the adaptive algorithms, the value of the momentum (learning step) is managed and updated by the algorithm. The impact of having a sub-optimal learning rate and divergence can be eliminated by modern optimizers.

We ran one experiment using both the Adam and RMS-prop algorithms to identify the best optimizer for our network. In Figure 4.7, we compared the performance of these two algorithms on dataset FD001 and, based on 10 experiments and by applying the learning rate of $1e-3$ which we obtained from the previous section. According to the result, the Adam optimizer leads to better performance and acceptable training time (less than RMS-prop). We also observed a dramatic fall in score value if we consider the first early epochs and those that come at the end. Based on the characteristic of the scoring function of [90], we can see that a significant improvement was obtained from a score of 4000 at early epochs to the final score of 400.



(a) Adam RMSE

(b) RMS-prop RMSE

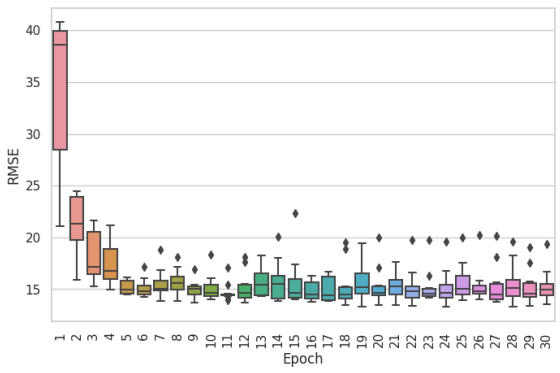
(c) Adam Score

(d) RMS-prop Score

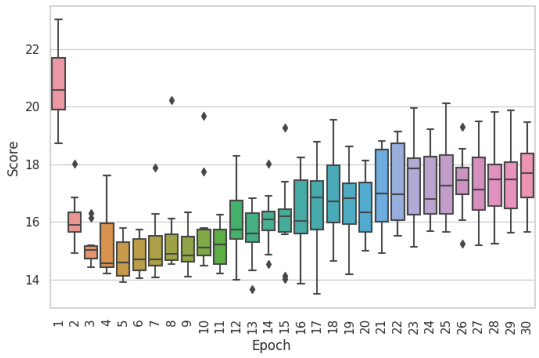
Figure 4.7: Illustration of RUL Prediction Performance over Epochs Using Adam vs. RMS-prop Optimizer on FD001, with Learning Rate of $1e-3$ and Based on 10 Experiments

4.4.3 Normalization

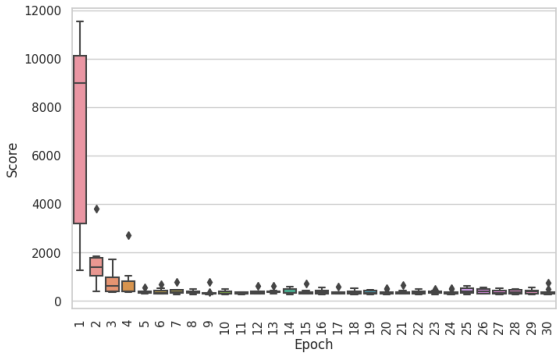
In Figure 4.8, we compare the performance of two normalization techniques, min-max and Z-score, also known as standardization, on dataset FD001, based on 10 experiments and by applying the learning rate of $1e-3$ which we obtained from Section 4.4.1. According to the results, Min-max normalization outperforms Z-score normalization with our settings. (scaling the input data within the range of [0, 1] and no outliers in the datasets).



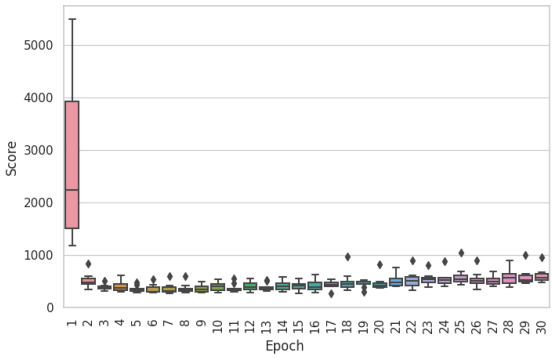
(a) Min-max RMSE



(b) Z-score RMSE



(c) Min-max Score

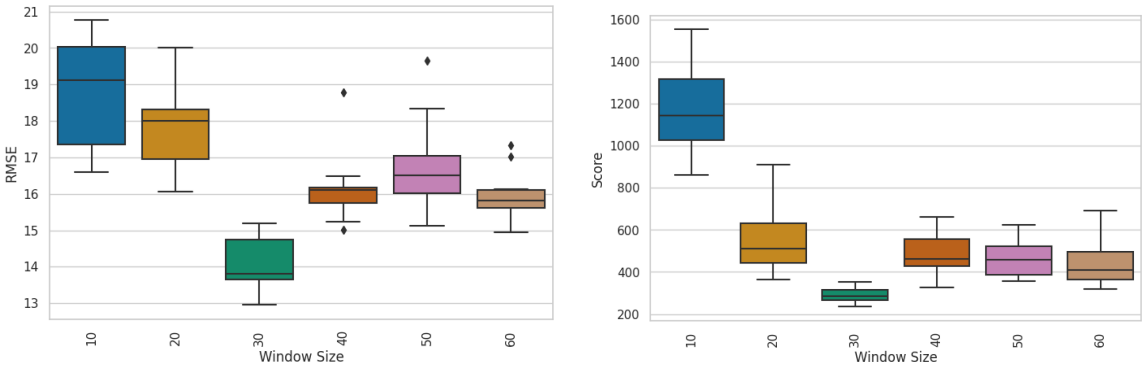


(d) Z-score Score

Figure 4.8: Illustration of RUL Prediction Performance over Epochs Using Min-max vs. Z-score as the Normalization Method on FD001, with Learning Rate of $1e-3$ and Based on 10 Experiments

4.4.4 Time Window Size

As discussed in Section 3.7, the time window technique can help increase the performance and improve the model's capability in predicting time-series data. In Figure 4.9, we show an overview of the network performance for various time window sizes. The overall trend in the figure is the same for all performance metrics. We considered window sizes of 10, 20, 30, 40, 50, and 60 for the experiments. As illustrated in Figure 4.9, by increasing the window size at the beginning due to the fact that we increase the size of the information input, the performance starts to show positive moves. But after increasing the size continuously, above a certain threshold, we see a negative impact of too much information on the performance. The figure depicts this trend for dataset FD001. A quite similar trend was also found on the FD004 dataset as well. Based on the FD004 dataset's more complex characteristics, the degradation in the performance is not as significant as FD001. We believe that this is because as data gets more complex, the level of information the network needs also increases. We use a window size of 30 for both datasets based on the results similar to what is suggested in [59].



(a) RMSE (b) Score

Figure 4.9: Illustration of RUL Prediction Performance for Various Time Window Sizes on FD001, Based on 10 Experiments

4.5 Comparison with State-of-the-Art

In this section, we compare our proposed architecture to other proposed models in the field. To verify the effectiveness of the proposed approach, we first performed an initial test on the training data of FD001, which is split for training and testing. A comparison has been made between the proposed approach and some widely used benchmark approaches, including DBN [59], RF [59], CNN [54], and LSTM [57] that all use RMSE and the scoring function of [90] as their evaluation metrics. We have also compared the proposed approach with state-of-the-art approaches on the testing data of FD001 and FD004. Table 4.3 shows the experimental results of the suggested approach and several state-of-the-art approaches on the two datasets. Overall, all the methods output better performance on the FD001 in comparison to the FD004. This is due to the FD001's simplicity, having only one operation condition and one fault type. Furthermore, the FD004 has 248 engines for testing, which is significantly more than the FD001. As a result, the scores that are sums of all the engines of the FD004 and FD001 are of varying magnitude.

Table 4.3: Comparison of the Experimental Results on FD001 and FD004 Sub-datasets with State-of-the-arts

	Criterion	**RF** [59]	**CNN** [54]	**LSTM** [57]	**DBN** [59]	**Proposed**
FD001	RMSE	17.91	18.45	15.42	15.21	13.27
	Score	479.75	1286.7	410.60	417.59	270.84
FD004	RMSE	31.12	29.16	29.12	29.88	25.16
	Score	46567.63	7886.4	12551.44	7954.51	5065.06

4.6 Discussion

The main goals of this research were first to show the impact of proper feature selection methods on the results of the neural network models and how it can improve the accuracy and, at the same time, lower the prediction time by reducing the dimensionality of the data. Many classic to modern dimensionality reduction techniques perform poorly in the final outputs. So, more advanced and complex approaches must be applied to the features to find an improving trend. Although recent models can deal with high dimensionality, we must still maintain the other advantage of that due to these advancements. The model's complexity can be lowered, which is one factor that can enhance such selected features. When using more complex deep learning architecture, reducing the input features can dramatically decrease the training time of the model.

The other goal was to show how we can modify the standard LSTM networks and take advantage of the attention mechanism in RUL prediction, which is a time series sequence analysis. As of the time of writing of this thesis, the MS-CNN deep learning architecture proposed by [94] had the highest performance, which is able to obtain an RMSE of 11.44 while in the best run, our proposed model could reach an RMSE of 12.71. This shows that the performance of the proposed model is not as strong as the hybrid and ensemble methods published in recent years. We can discuss this further by considering two different points. According to our goal and objectives, we wanted to show that the impact of the attention mechanism, even on a regular neural network architecture, can dramatically improve performance. So having this hypothesis on hand, we can apply the findings here to more advanced hybrid stacked architecture and output better results. The other point is that all hybrid and ensemble-based methods generally sacrifice the complexity and expensiveness of computational time and resources for the advancement of current systems. This may not be noticeable, but feature selection and how to feed the data to the network are essential in the domain.

Chapter 5

Conclusion and Future Perspectives

This thesis addressed the problem of system prognostic health monitoring (PHM), specifically, focusing on predicting the remaining useful life (RUL) of a system. We proposed a neural network model based on the attention mechanism to help us, first, better understand the data space, and also to evaluate the importance of each sequence as the output of our proposed LSTM layer. Experiments and studies were conducted to achieve our objectives, and their outcomes are presented. In this chapter, we will review our contributions and future work.

5.1 Contributions

The model proposed in this work, allows us to predict the health status of a system without requiring the knowledge and experience of an expert in the domain. We showed the importance of targeting each component's contribution in a predictive maintenance system and how we can extract the most informative ones for a better view of the whole system's status. We experimented with various methods to prepare the raw data for the experiments and explained how to evaluate and measure different settings' success. We investigated

how different machine learning methods obtain their outputs and how we can take advantage of them based on the nature of the problem we focused on. Finally, we described how we incorporated the attention mechanism into the neural network pipeline and showed it resulted in noticeable improvements using the RMSE and proposed score function [90] metrics.

5.2 Future Perspectives

For future work, we suggest the following:

- Our work focused on a single machine learning methodology, to increase the performance of RUL prediction, one area to explore is the development of hybrid methodologies that integrate several techniques, such as physics-based models and machine learning algorithms. Also, more research is essential to include RUL prediction in maintenance decision-making. In order to achieve this, it is important to develop systems for decision support that can optimize predictive maintenance and reduce downtime by considering the predicted RUL and cutting repair costs.

- Our work Assumed that the RUL degradation trend is a piece-wise linear function. In a real-life scenario, this may not be the case. One potential future work would be to deploy different degradation trends or even try to find patterns similar to the true degradation in different sensor readings.

- To improve the performance and decrease both training and test time consumption of models, more advanced and powerful feature selection methods could be applied to the C-MAPSS datasets to see if they can improve the neural network's performance by reducing the dimensionality further. This would allow the inclusion of only the features with higher contributions to the system that would result in better performance. It is worth saying that such improvements would also reduce the impact of

biased data on the training procedure and the final model.

- Finally, this thesis has assumed that the datasets were balanced. In real-life scenarios, the data exhibit a high level of imbalance. This, on its own, is an important topic to focus on, that is necessary to address in order to deploy any RUL predictor systems in the industry.

Bibliography

[1] Jonathan Trout. While predictive/preventive maintenance is still king, maintenance personnel are reluctant to use internet-based maintenance, Nov 2019. https://www.reliableplant.com/Read/31707/predictive-maintenance-survey-2019, Accessed: 2022-06-18.

[2] Ranganath Kothamasu, Samuel Huang, and William VerDuin. System health monitoring and prognostics – A review of current paradigms and practices. *Handbook of Maintenance Management and Engineering*, 28:1012–1024, 07 2006.

[3] Jialin Li, X. Li, and D. He. A directed acyclic graph network combined with CNN and LSTM for remaining useful life prediction. *IEEE Access*, 7:75464–75475, 2019.

[4] Gil-Yong Lee, Mincheol Kim, Ying-Jun Quan, Min-Sik Kim, Thomas Kim, Hae-Sung Yoon, Sangkee Min, Dong-Hyeon Kim, Jeong-Wook Mun, Jin Oh, In Choi, Chung-Soo Kim, Won-Shik Chu, Jinkyu Yang, Binayak Bhandari, Choon-Man Lee, Jeong-Beom Ihn, and Sung-Hoon Ahn. Machine health management in smart factory: A review. *Journal of Mechanical Science and Technology*, 32:987–1009, 03 2018.

[5] Jay Lee, Fangji Wu, Wenyu Zhao, Masoud Ghaffari, Linxia Liao, and David Siegel. Prognostics and health management design for rotary machinery systems-reviews, methodology and applications. *Mechanical Systems and Signal Processing*, 42(1):314–334, 2014.

[6] Ebru Turanoğlu Bekar, Per Nyqvist, and Anders Skoogh. An intelligent approach for data pre-processing and analysis in predictive maintenance with an industrial case study. *Advances in Mechanical Engineering*, 2020.

[7] R. Keith Mobley. Contents. In *An Introduction to Predictive Maintenance (Second Edition)*, Plant Engineering, pages v–xii. Butterworth-Heinemann, Burlington, second edition, 2002.

[8] A. Abu-Hanna and Peter J. Lucas. Prognostic models in medicine: AI and statistical approaches. *Methods of information in medicine*, 40:1–5, 04 2001.

[9] W. Skamarock, J. Klemp, and Jimy Dudhia. Prototypes for the WRF (weather research and forecasting) model. In *Preprints, Ninth Conf. Mesoscale Processes, J11–J15, Amer. Meteorol. Soc., Fort Lauderdale, FL*, 01 2001.

[10] Victor Giurgiutiu. Tuned lamb wave excitation and detection with piezoelectric wafer active sensors for structural health monitoring. *Journal of Intelligent Material Systems and Structures*, 16, 04 2005.

[11] Claudio Sbarufatti, Matteo Corbetta, Andrea Manes, and Marco Giglio. Sequential monte-carlo sampling based on a committee of artificial neural networks for posterior state estimation and residual lifetime prediction. *International Journal of Fatigue*, 83, 06 2015.

[12] Arkadiusz Gertych, Zaneta Swiderska, Zhaoxuan Ma, Nathan Ing, Tomasz Markiewicz, Szczepan Cierniak, Hootan Salemi, Samuel Guzman, Ann Walts, and Beatrice Knudsen. Convolutional neural networks can accurately distinguish four histologic growth patterns of lung adenocarcinoma in digital slides. *Scientific Reports*, 9:1483, 02 2019.

[13] Frank L Greitzer and Ronald A Pawlowski. Embedded prognostics health monitoring. *Proceedings of the 48th International Instrumentation Symposium, May 5-9 2002, San Diego, CA*, 5 2002.

[14] Brian Bole, Chetan Kulkarni, and Matthew Daigle. Adaptation of an electrochemistry-based Li-Ion battery model to account for deterioration observed under randomized use. *PHM 2014 - Proceedings of the Annual Conference of the Prognostics and Health Management Society 2014*, 09 2014.

[15] Oleg Gusikhin, Nestor Rychtyckyj, and Dimitar Filev. Intelligent systems in the automotive industry: Applications and trends. *Knowl. Inf. Syst.*, 12:147–168, 07 2007.

[16] Jianhui Luo, M. Namburu, K. Pattipati, Liu Qiao, M. Kawamoto, and S. Chigusa. Model-based prognostic techniques [maintenance applications]. In *Proceeding of the AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference*, pages 330–340, 2003.

[17] Dragan Djurdjanovic, Jay Lee, and Jun Ni. Watchdog agent — An infotronics-based prognostics approach for product performance degradation assessment and prediction. *Advanced Engineering Informatics*, 17:109–125, 07 2003.

[18] G. Vachtsevanos and P. Wang. Fault prognosis using dynamic wavelet neural networks. In *2001 IEEE Autotestcon Proceedings. IEEE Systems Readiness Technology Conference. (Cat. No.01CH37237)*, pages 857–870, 2001.

[19] Dong Wang, Kwok-Leung Tsui, and Qiang Miao. Prognostics and health management: A review of vibration based bearing and gear health indicators. *IEEE Access*, 6:665–676, 2018.

[20] Emmanuel Ramasso and Abhinav Saxena. Review and Analysis of Algorithmic Approaches Developed for Prognostics on CMAPSS Dataset. In *Annual Conference*

of the Prognostics and Health Management Society 2014., Fort Worth, TX, USA., United States, September 2014.

[21] Hatem Elattar, Hamdy Elminir, and Alaa el-din Riad. Prognostics: A literature review. *Complex & Intelligent Systems*, 2, 06 2016.

[22] Advanced Technology Services Inc. Downtime costs auto industry $ 22k/minute - survey. `https://news.thomasnet.com/companystory/downtime-costs-auto-industry-22k-minute-survey-481017`, Accessed: 2022-06-18.

[23] Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. *Proceeding of the International Conference on Prognostics and Health Management*, 10 2008.

[24] Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and Mérouane Debbah. Artificial neural networks-based machine learning for wireless networks: A tutorial. *IEEE Communications Surveys & Tutorials*, 21(4):3039–3071, 2019.

[25] Andrew S. Tenney, Mark N. Glauser, Christopher J. Ruscher, and Zachary P. Berger. Application of artificial neural networks to stochastic estimation and jet noise modeling. *AIAA Journal*, 58(2):647–658, 2020.

[26] Fjodor van Veen. The neural network zoo, Apr 2019. `https://www.asimovinstitute.org/neural-network-zoo/`, Accessed: 2022-04-10.

[27] Stephan Trenn. Multilayer perceptrons: Approximation order and necessary number of hidden units. *IEEE Transactions on Neural Networks*, 19(5):836–844, 2008.

[28] Shaode Yu, shibin Wu, Lei Wang, Fan Jiang, Yaoqin Xie, and Leida Li. A shallow convolutional neural network for blind image sharpness assessment. *PLOS ONE*, 12:e0176632, 05 2017.

[29] Guoxing Lan, Qing Li, and Nong Cheng. Remaining useful life estimation of turbofan engine using lstm neural networks. In *Proceeding of the 2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*, pages 1–5, 2018.

[30] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[32] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[33] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML*, page 2048 – 2057, 2016.

[34] Wenbin Du, Yali Wang, and Yu Qiao. Recurrent spatial-temporal attention network for action recognition in videos. *Proceeding of the IEEE Transactions on Image Processing*, 27(3):1347–1360, 2018.

[35] Wu Yuankai, Huachun Tan, Bin Ran, and Zhuxi Jiang. A hybrid deep learning based traffic flow prediction method and its understanding. *Transportation Research Part C: Emerging Technologies*, 90, 05 2018.

[36] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun,

editors, *Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 7-9*, 2015.

[37] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[38] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding, 2017.

[39] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas, November 2016. Association for Computational Linguistics.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[41] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations, 2018.

[42] Mingzhou Xu, Derek F. Wong, Baosong Yang, Yue Zhang, and Lidia S. Chao. Leveraging local and global patterns for self-attention networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3069–3075, Florence, Italy, July 2019. Association for Computational Linguistics.

[43] Yagmur Gizem Cinar, Hamid Mirisaee, Parantapa Goswami, Eric Gaussier, Ali Aït-Bachir, and Vadim Strijov. Position-based content attention for time series forecasting with sequence-to-sequence rnns. In Derong Liu, Shengli Xie, Yuanqing Li, Dongbin Zhao, and El-Sayed M. El-Alfy, editors, *Neural Information Processing*, pages 533–544, Cham, 2017. Springer International Publishing.

[44] Han Shi, Jiahui Gao, Xiaozhe Ren, Hang Xu, Xiaodan Liang, Zhenguo Li, and James T. Kwok. Sparsebert: Rethinking the importance analysis in self-attention, 2021.

[45] Zhigang Tian. An artificial neural network approach for remaining useful life prediction of equipments subject to condition monitoring. In *Proceedings of the 2009 8th International Conference on Reliability, Maintainability, and Safety*, pages 143–148, 2009.

[46] David J. C. MacKay. Comparison of approximate methods for handling hyperparameters. *Neural Computation*, 11(5):1035–1068, 1999.

[47] Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint*, arXiv, 07 2012.

[48] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[49] Man Shan Kan, Andy C.C. Tan, and Joseph Mathew. A review on prognostic techniques for non-stationary and non-linear rotating systems. *Mechanical Systems and Signal Processing*, 62-63:1–20, 2015.

[50] Ning-Cong Xiao, Kai Yuan, and Hongyou Zhan. System reliability analysis based on dependent kriging predictions and parallel learning strategy. *Reliability Engineering & System Safety*, 218:108083, 2022.

[51] Jianhui Luo, M. Namburu, K. Pattipati, Liu Qiao, M. Kawamoto, and S. Chigusa. Model-based prognostic techniques [maintenance applications]. In *Proceedings AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference.*, pages 330–340, 2003.

[52] Chenglong Zhang, Shifei Ding, Yuting Sun, and Zichen Zhang. An optimized support vector regression for prediction of bearing degradation. *Applied Soft Computing*, 113:108008, 2021.

[53] Jason Deutsch and David He. Using deep learning-based approach to predict remaining useful life of rotating components. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(1):11–20, 2018.

[54] G. Sateesh Babu, Peilin Zhao, and Xiaoli Li. Deep convolutional neural network based regression approach for estimation of remaining useful life. In *Database Systems for Advanced Applications*, 2016.

[55] Jun Zhu, Nan Chen, and Weiwen Peng. Estimation of bearing remaining useful life based on multiscale convolutional neural network. *Proceeding of the IEEE Transactions on Industrial Electronics*, 66(4):3208–3216, 2019.

[56] Xiang Li, Wei Zhang, and Qian Ding. Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction. *Reliability Engineering & System Safety*, 182, 11 2018.

[57] Shuai Zheng, Kosta Ristovski, Ahmed Farahat, and Chetan Gupta. Long short-term memory network for remaining useful life estimation. In *Proceeding of the 2017*

IEEE International Conference on Prognostics and Health Management (ICPHM), pages 88–95, 2017.

[58] Jianjing Zhang, Peng Wang, Ruqiang Yan, and Robert Gao. Long short-term memory for machine remaining life prediction. *Journal of Manufacturing Systems*, 48, 06 2018.

[59] Chong Zhang, Pin Lim, A. K. Qin, and Kay Chen Tan. Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics. *Proceeding of the IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2306–2318, 2017.

[60] Mei Yuan, Yuting Wu, and Li Lin. Fault diagnosis and remaining useful life estimation of aero engine using LSTM neural network. In *Proceeding of the 2016 IEEE International Conference on Aircraft Utility Systems (AUS)*, pages 135–140, 2016.

[61] Yuting Wu, Mei Yuan, Shaopeng Dong, Li Lin, and Yingqi Liu. Remaining useful life estimation of engineered systems using vanilla LSTM neural networks. *Neurocomputing*, 275, 05 2017.

[62] André Listou Ellefsen, Emil Bjørlykhaug, Vilmar Æsøy, Sergey Ushakov, and Houxiang Zhang. Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture. *Reliability Engineering & System Safety*, 183:240–251, 2019.

[63] Meidi Sun, Hui Wang, Ping Liu, Shoudao Huang, and Peng Fan. A sparse stacked denoising autoencoder with optimized transfer learning applied to the fault diagnosis of rolling bearings. *Measurement*, 146:305–314, 2019.

[64] Ramin M. Hasani, Guodong Wang, and Radu Grosu. An automated auto-encoder correlation-based health-monitoring and prognostic method for machine bearings. *CoRR*, abs/1703.06272, 2017.

[65] Jehn-Ruey Jiang, Juei-En Lee, and Yi-Ming Zeng. Time series multiple channel convolutional neural network with attention-based long short-term memory for predicting bearing remaining useful life. *Sensors*, 20(1), 2020.

[66] Henghui Jia, Weijie Dong, Hongtao Liu, Junjie Yan, and Jie Gao. Hybrid RUL prediction model based on support vector regression and degradation law for gas turbine. *Journal of Intelligent Manufacturing*, 32(4):1017–1033, 2021.

[67] Lei Ren, Jiabao Dong, Xiaokang Wang, Zihao Meng, Li Zhao, and M. Jamal Deen. A data-driven auto-cnn-lstm prediction model for lithium-ion battery remaining useful life. *IEEE Transactions on Industrial Informatics*, 17(5):3478–3487, 2021.

[68] Fangfang Wang, Yilin Ma, Pengpeng Zhang, Xuefeng Chen, and Jie Zhang. An ensemble model based on multiple regression models for remaining useful life prediction of gearbox. *Measurement*, 141:261–270, 2019.

[69] Chang-Hua Hu, Hong Pei, Xiao-Sheng Si, Dang-Bo Du, Zhe-Nan Pang, and Xi Wang. A prognostic model based on dbn and diffusion process for degrading bearing. *IEEE Transactions on Industrial Electronics*, 67(10):8767–8777, 2020.

[70] Ali Al-Dulaimi, Soheil Zabihi, Amir Asif, and Arash Mohammadi. Hybrid deep neural network model for remaining useful life estimation. In *Proceeding of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3872–3876, 2019.

[71] Hui Liu, Zhenyu Liu, Weiqiang Jia, and Xianke Lin. A novel deep learning-based encoder-decoder model for remaining useful life prediction. In *Proceeding of the 2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[72] Shaashwat Agrawal, Sagnik Sarkar, Gautam Srivastava, Praveen Reddy, and Thippa Gadekallu. Genetically optimized prediction of remaining useful life. *Sustainable Computing: Informatics and Systems*, 31:100565, 05 2021.

[73] Penghua Li, Zijian Zhang, Qingyu Xiong, Baocang Ding, Jie Hou, Dechao Luo, Yujun Rong, and Shuaiyong Li. State-of-health estimation and remaining useful life prediction for the lithium-ion battery based on a variant long short term memory neural network. *Journal of Power Sources*, 459:228069, 05 2020.

[74] Yong Yu, Changhua Hu, Xiaosheng Si, Jianfei Zheng, and Jianxun Zhang. Averaged Bi-LSTM networks for RUL prognostics with non-life-cycle labeled dataset. *Neurocomputing*, 402:134–147, 2020.

[75] Sweta Bhattacharya, Praveen Reddy, Iyapparaja Meenakshisundaram, Thippa Gadekallu, Sparsh Sharma, Mustufa Abidi, and Mohammed Alkahtani. Deep neural networks based approach for battery life prediction. *CMC -Tech Science Press-*, 69:2599–2615, 07 2021.

[76] Dean Frederick, Jonathan DeCastro, and Jonathan Litt. User's guide for the commercial modular aero-propulsion system simulation (C-MAPSS). *NASA Technical Manuscript*, 2007–215026, 01 2007.

[77] Abhinav Saxena and Kai Goebel. C-MAPSS data set. *NASA Ames Prognostics Data Repository*, 01 2008.

[78] Leto Peel. Data driven prognostics using a kalman filter ensemble of neural network models. In *Proceeding of the 2008 International Conference on Prognostics and Health Management*, pages 1–6, 2008.

[79] Felix O. Heimes. Recurrent neural networks for remaining useful life estimation. In *Proceeding of the 2008 International Conference on Prognostics and Health Management*, pages 1–6, 2008.

[80] Mohamed Sayah, Djillali Guebli, Noureddine Zerhouni, and Zeina Al Masry. Towards distribution clustering-based deep LSTM models for RUL prediction. In *Proceeding of the 2020 Prognostics and Health Management Conference (PHM-Besançon)*, pages 253–256, 2020.

[81] Che-Sheng Hsu and Jehn-Ruey Jiang. Remaining useful life estimation using long short-term memory deep learning. In *Proceeding of the 2018 IEEE International Conference on Applied System Invention (ICASI)*, pages 58–61, 2018.

[82] Jackson Cornelius, Blake Brockner, Seong Hyeon Hong, Yi Wang, Kapil Pant, and John Ball. Estimating and leveraging uncertainties in deep learning for remaining useful life prediction in mechanical systems. In *Proceeding of the 2020 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 1–8, 2020.

[83] Lim Pin, Chi Keong Goh, and Kay Chen Tan. A time window neural network based framework for remaining useful life estimation. In *Proceeding of the 2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1746–1753, 2016.

[84] Ying Peng, Ming Dong, and Ming Jian Zuo. Current status of machine prognostics in condition-based maintenance: A review. *The International Journal of Advanced Manufacturing Technology*, 50(1):297–313, 2010.

[85] Lim Pin, Chi-Keong Goh, K.C. Tan, and P. Dutta. Estimation of remaining useful life based on switching kalman filter neural network ensemble. *In Proceeding of the Annual Conference of the Prognostics and Health Management Society (PHM)*, pages 2–9, 01 2014.

[86] Caifeng Zheng, Weirong Liu, Bin Chen, Dianzhu Gao, Yijun Cheng, Yingze Yang, Xiaoyong Zhang, Shuo Li, Zhiwu Huang, and Jun Peng. A data-driven approach for remaining useful life prediction of aircraft engines. In *Proceeding of the 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 184–189, 2018.

[87] Faisal Khan, Ömer Eker, Atif Khan, and Wasim Orfali. Adaptive degradation prognostic reasoning by particle filter with a neural network degradation model for turbofan jet engine. *Data*, 3:49, 11 2018.

[88] Hai-Kun Wang, Yi Cheng, Ke Song, and Thippa Reddy G. Remaining useful life estimation of aircraft engines using a joint deep learning model based on tcnn and transformer. *Intell. Neuroscience*, 2021, jan 2021.

[89] Tianyi Wang, Jianbo Yu, David Siegel, and Jay Lee. A similarity-based prognostics approach for remaining useful life estimation of engineered systems. In *Proceedings of the 2008 International Conference on Prognostics and Health Management*, pages 1–6, 2008.

[90] Abhinav Saxena, Jose Celaya, Edward Balaban, Kai Goebel, Bhaskar Saha, Sankalita Saha, and Mark Schwabacher. Metrics for evaluating performance of prognostic techniques. In *2008 International Conference on Prognostics and Health Management*, pages 1–17, 2008.

[91] Yaohua Deng, Chengwang Guo, Zilin Zhang, Linfeng Zou, Xiali Liu, and Shengyu Lin. An attention-based method for remaining useful life prediction of rotating machinery. *Applied Sciences*, 13(4), 2023.

[92] Ibrahem Kandel and Mauro Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, 6(4):312–315, 2020.

[93] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[94] Han Li, Wei Zhao, Yuxi Zhang, and Enrico Zio. Remaining useful life prediction using multi-scale deep convolutional neural network. *Applied Soft Computing*, 89:106 – 113, 2020.