ABSTRACT

Predicting the life condition of a component is so crucial for industries that have intent to grow in a fast-paced technological environment. The area of predictive maintenance has taken a lot of prominence in the last couple of years due to various reasons. With new algorithms and methodologies growing across different learning methods, it has remained a challenge for industries to adopt which method is fit, robust and provide the most accurate detection. Fault detection is one of the critical components of predictive maintenance; it is very much needed for industries to detect faults early and accurately. Predicting the life condition of a component is so crucial for industries that have intent to grow in a fast-paced technological environment. Recent studies on predictive maintenance help industries to create an alert before the components are corrupted. Prediction of component failures, companies have a chance to sustain their operations efficiently while reducing their maintenance cost by repairing components in advance. To predict and be ahead of all these scenarios we imply Artificial Intelligence (AI) and Deep Learning using long short-term memory (LSTM) neural networks to predict when an aircraft engine might require to be serviced or replaced. In this paper, Long Short-Term Memory (LSTM) networks have been performed to predict the current situation of an engine. LSTM model deals with sequential input data. The training process of LSTM networks has been performed on large-scale data processing engine with high performance. For these predictions, we require a dataset of previous aircraft historical data having 21 sensor values of each aircraft. Aircraft have three different settings = set1, set2, set3 so that we can manipulate the data and find out the relation/trend in the data so that our system gets capable of predicting remaining useful life (RUL) of an aircraft engine.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

S.NO	ACRONYM	DEFINITION
01.	CNN	Convolutional Neural Networks
02.	ML	Machine Learning
03.	RNN	Recurrent Neural Networks
04.	DL	Deep Learning
05.	MAPSS	Modular Aero-Propulsion System Simulation
06.	RUL	Remaining Useful Life
07.	LSTM	Long Short-Term Memory
08.	NASA	National Aeronautics and Space Administration
09.	PDM	Predictive Maintenance

LIST OF SYMBOLS

SYMBOLS	**DEFINITION**	**PAGE NO.**
^	Caret	17
π	Pie	19
φ	Phi	22
Σ	Sigma	22

CHAPTER 1 - INTRODUCTION

1.1 INTRODUCTION

As technology is developing rapidly, companies have the ability to observe the health of engine components and constructed systems through collecting signals from sensors. According to the output of IoT sensors, companies can build systems to predict the conditions of components. Practically the components are required to be maintained or replaced before the end of life in performing their assigned task. Predicting the life condition of a component is so crucial for industries that have intent to grow in a fast-paced technological environment. Recent studies on predictive maintenance help industries to create an alert before the components are corrupted. Thanks to the prediction of component failures, companies have a chance to sustain their operations efficiently while reducing their maintenance cost by repairing components in advance. Since maintenance affects production capacity and the service quality directly, optimized maintenance is the key factor for organizations to have more revenue and stay competitive in developing industrialized world.

With the aid of a well-designed prediction system for understanding the current situation of an engine, components could be taken out of active service before a malfunction occurs. With the help of inspection, effective maintenance extends component life, improves equipment availability and keeps components in a proper condition while reducing costs. Real-time data collected from sensors is a great source to model component deteriorations. Markov Chain models, Survival Analysis, Optimization algorithms and several machine learning approaches have been implemented to model predictive maintenance.

The system reliability is one of the most critical points for engineering operations. Failure of some parts of the system could affect all of the operations. Turbine engines, power supplies and batteries are typical instances that could cause operation failure. To avoid break down condition, some or all parts of the system should be well maintained. In common maintenance strategies, part of a system is repaired when failure is observed.

The issue of remaining useful life (RUL) prediction has already become a quite interesting topic in an industrial product. The data-driven RUL prediction has been applied to the current research by taking advantage of a long-short term memory (LSTM) -recurrent neural network (RNN) approach. This means that even in a specified long-short term memory bound and limited available data sets, the RUL predictions can also improve the equipment capacity. By collecting the sensor parameters from the National Aeronautics and Space Administration (NASA) jet engines, the capability of this approach has been demonstrated. An appropriate selection, that is suitable for the variable measurement, has been used to feed LSTM-RNNs with the most useful RUL labels. The analysis results illustrate that, compared with a traditional statistical model, the development prediction approach is likely to provide an accurate prediction for the remaining useful life of the equipment that can be meaningful to maintenance schemes.

During the past decades, Aircraft engine was been developed with a minimum number of sensors as there was no requirement of various other sensor value which significance the engine condition of an Aircraft. Now having all the new 21 sensor embedded inside an engine of an aircraft makes us do predictive maintenance so that we save a lot of time and energy to avoid the necessity of doing unnecessary maintenance service. These sensors fitted the Aircraft Engine provide a huge amount of previous data that shows the real condition of the engine. These huge amounts of data may be stored in Hard Disk of an aircraft engine or on the servers located in closed making it easier to locate and work with it whenever it is required. Hence by reducing the manual time to go work with the engines, so the maintenance service can be placed locally and can manipulate with the data stored in the closed and do predictive maintenance as and when required. The deep learning approach of implementing Recurrent neural networks (RNN) and using long short term memory (LSTM) technique makes us predict the data set with timestamps which means the present data looks fifty-time stamps in the past to predict the current remaining useful life (RUL) of the Aircraft engine and hence making it to be very accurate.

1.2 PROJECT SCOPE:

Fault detection is one of the preliminary analytics for predictive maintenance. Hence, detecting the fault accurately is regarded as important. This work is currently performed for vibration data. The scope of this research can be extended out to other physics-based parameters and combination of these parameters. It would also be interesting to observe the detection accuracy for bigger sample size and multiple fault states.

While getting technology to work may be central to PdM 4.0, the scope of implementing it is far wider. Companies should also pay attention to organisational dimensions, and ensure the project management and change management skills needed for a successful PdM 4.0 implementation. This chapter sets out an implementation approach for PdM 4.0, which includes the technological and organisational aspects that companies must address to make the most of PdM 4.0.

1.3 PROJECT PURPOSE:

The purpose of the paper is twofold. First of all, we perform an analysis which shows how the hyperparameters can affect the model with the final goal to discover possible correlations between them and the model performance. To do that, we used a dataset containing the full history of a set of engines until their fault provided by NASA that can be considered a standard to train and test models for the RUL prediction.

Predictive maintenance techniques are designed to help determine the condition of in-service equipment to estimate when maintenance should be performed. This approach promises cost savings over routine or time-based preventive maintenance because tasks are performed only when warranted. Thus, it is regarded as condition-based maintenance carried out as suggested by estimations of the degradation state of an item. The main promise of predictive maintenance is to allow convenient scheduling of corrective maintenance and to prevent unexpected equipment failures. The key is "the right infer equipment lifetime, increased plant safety, fewer accidents with a negative impact on the environment, and optimized spare parts handling. Predictive maintenance differs from preventive maintenance because it relies on the actual condition of equipment, rather than average or expected life statistics, to predict when maintenance will be required. Some of the main components that are necessary for implementing predictive maintenance are data collection and pre-processing, early fault detection, fault detection, time to failure prediction, maintenance scheduling and resource optimization. Predictive maintenance has also been considered to be one of the driving forces for improving productivity and one of the ways to achieve "just-in-time" in manufacturing.

1.4 PROJECT FEATURES:

Predictive maintenance evaluates the condition of equipment by performing periodic or continuous equipment condition monitoring. The ultimate goal of the approach is to perform maintenance at a scheduled point in time when the maintenance activity is most cost-effective and before the equipment loses performance within a threshold. This results in a reduction in unplanned downtime costs because of failure where for instance costs can be in the hundreds of thousands per day depending on the industry. In energy production in addition to the loss of revenue and component costs, fines can be levied for non-delivery increasing costs even further. This is in contrast to time- and/or operation count-based maintenance, where a piece of equipment gets maintained whether it needs it or not. Time-based maintenance is labour intensive, ineffective in identifying problems that develop between scheduled inspections, and so is not cost-effective. The "predictive" component of predictive maintenance stems from the goal of predicting the future trend of the equipment's condition. This approach uses principles of statistical process control to determine at what point in the future maintenance activities will be appropriate. Most predictive inspections are performed while equipment is in service, thereby minimizing disruption of normal system operations. Adoption of predictive maintenance can result in substantial cost savings and higher system reliability. Reliability-centred maintenance emphasizes the use of predictive maintenance techniques in addition to traditional preventive measures. When properly implemented, it provides companies with a tool for achieving the lowest asset net present costs for a given level of performance and risk.

Based on having high accuracy this technique can be greatly implied and utilize providing better service to the Aircraft industry. These sensors fitted the Aircraft Engine provide a huge amount of previous data that shows the real condition of the engine. These huge amounts of data may be stored in Hard Disk of an aircraft engine or on the servers located in closed making it easier to locate and work with it whenever it is required. Hence by reducing the manual time to go work with the engines, so the maintenance service can be placed locally and can manipulate with the data stored in the closed and do predictive maintenance as and when required. The deep learning approach of implementing Recurrent neural networks(RNN) and using long short term memory(LSTM) technique makes us predict the data set with timestamps which means the present data looks many timestamps in the past to predict the current remaining useful life(RUL) of the Aircraft engine and hence making it to be highly accurate.

CHAPTER 2 - LITERATURE REVIEW & PROBLEM IDENTIFICATION

The primary goal of Predictive Maintenance is to reduce the cost of a product or service and to have a competitive advantage in the market to survive. Today business analytics are embedded across Predictive Maintenance to realize the need for it and to make appropriate decisions. Business analytics can be viewed in three different prospective

(i) Descriptive analytics
(ii) Predictive analytics and
(iii) Prescriptive analytics.

Descriptive analytics is a process of answering questions like what happened in the past? This is done by analysing historical data and summarizing them in charts. In maintenance, this step is performed using control charts.

Predictive analytics is an extension to descriptive analytics where historical data is analysed to predict future outcomes. In maintenance, it is used to predict the type of failure and time to complete failure.

Prescriptive analytics is a process of optimization to identify the best alternatives to minimize or maximize the objective. This also answers questions such as what can be done? In maintenance, this can be used to optimize the maintenance schedules to minimize the cost of maintenance.

In this paper, our primary focus will be on descriptive and predictive analytics to detect the faults. Predictive analytics has spread its applications into various applications such as railway track maintenance, vehicle monitoring, automotive subcomponents, utility systems, computer systems, electrical grids, aircraft maintenance, oil and gas industry, computational finance and many more. Fault detection is one of the concepts in predictive maintenance which is well accepted in the industry. Early Failure detection could potentially eliminate catastrophic machine failures.

2.1 PROBLEM IDENTIFICATION

Before the introduction of predictive maintenance, machinery was maintained by trained professionals who used to perform maintenance at regular intervals to avoid catastrophic chain failure of different machines. The problem with this approach means extended machine downtime affecting the production chain. Changing the parts even though they are in good condition also increases the maintenance costs. On the one hand, an unexpected failure can result in a devastating accident and financial losses for the company owing to the interaction behaviours among industrial equipment. On the other hand, early detection and prediction of a fault can prevent it from growing and eventually turning into critical problems. Hence, increasing attention has been paid to condition monitoring, fault diagnosis, and prognosis in modern industry. It is well known that machine faults can result in consequences that may range from a simple replacement of a cheap bearing to an accident that will cost millions in lost production, injuries, or pollution. It may also bother maintenance engineers to capture the trade-off between improving the system reliability and reducing the total maintenance cost simultaneously.

Electric machines are widely employed in a variety of industrial applications and electrified transportation systems. For certain applications, these machines may operate under unfavorable conditions, such as high ambient temperature, high moisture and overload, which can eventually result in motor malfunctions that lead to high maintenance costs, severe financial losses, and safety hazards. The malfunction of electric machines can be generally attributed to various faults of different categories, including drive inverter failures, stator winding insulation breakdown, bearing faults and air gap eccentricity.

2.2 PROBLEM DESCRIPTION

It could be clearly said that accurate estimation of engine condition based on sensors data has many benefits and advantages for the prognosis of the engine's current condition. In this paper, the data is about engine degradation simulation(C-MAPSS) provided by the Prognostics CoE at NASA

Ames has been used. Data set includes different combinations of operational conditions and fault modes in multivariate time series format. Data consists of three operational settings and 21 sensor measurements. Sensors are collecting data related temperature, engine pressure, fuel, coolant bleed. Details could be found in Table I.

Operational Settings	
Settings No	**Description**
1	Altitude
2	Mach number
3	Throttle resolver angle
Sensor Measurements	
Sensor No	**Description**
1	Total temperature at fan inlet (°R)
2	Total temperature at LPC outlet (°R)
3	Total temperature at HPC outlet (°R)
4	Total temperature at LPT outlet (°R)
5	Pressure at fan inlet (psia)
6	Total pressure in bypass-duct (psia)
7	Total pressure at HPC outlet(psia)
8	Physical fan speed (rpm)
9	Physical core speed (rpm)
10	Engine pressure ratio (P50/P2)
11	Engine pressure ratio (P50/P2)
12	Ratio of fuel flow to Ps30 (pps/psi)
13	Corrected fan speed (rpm)
14	Corrected core speed (rpm)
15	Bypass Ratio
16	Burner fuel-air ratio
17	Bleed Enthalpy
18	Demanded fan speed (rpm)
19	Demanded corrected fan speed (rpm)
20	HPT coolant bleed (lbm/s)
21	LPT coolant bleed (lbm/s)

Table: 2.1 Operational Settings

CHAPTER 3- METHODOLOGY

3.1 DEEP LEARNING

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It's achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labelled data and neural network architectures that contain many layers. In a word, accuracy. Deep learning achieves recognition accuracy at higher levels than ever before. This helps consumer electronics meet user expectations, and it is crucial for safety-critical applications like driverless cars. Recent advances in deep learning have improved to the point where deep learning outperforms humans in some tasks like classifying objects in images.

While deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful:

1. Deep learning requires large amounts of **labelled data**. For example, driverless car development requires millions of images and thousands of hours of video.
2. Deep learning requires substantial **computing power**. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.

3.1.1 Deep learning applications

Automated Driving: Automotive researchers are using deep learning to automatically detect objects such as stop signs and traffic lights. Also, deep learning is used to detect pedestrians, which helps decrease accidents.

- Aerospace and Defence: Deep learning is used to identify objects from satellites that locate areas of interest, and identify safe or unsafe zones for troops.

- Medical Research: Cancer researchers are using deep learning to automatically detect cancer cells. Teams at UCLA built an advanced microscope that yields a high-dimensional data set used to train a deep learning application to accurately identify cancer cells.

- Industrial Automation: Deep learning is helping to improve worker safety around heavy machinery by automatically detecting when people or objects are within an unsafe distance of machines.

- Electronics: Deep learning is being used in automated hearing and speech translation. For example, home assistance devices that respond to your voice and know your preferences are powered by deep learning applications. Most deep learning methods use **neural network** architectures, which is why deep learning models are often referred to as **deep neural networks**.

The term “deep” usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150.Deep learning models are trained by using large sets of labelled data and neural network architectures that learn features directly from the data without the need for manual feature extraction.

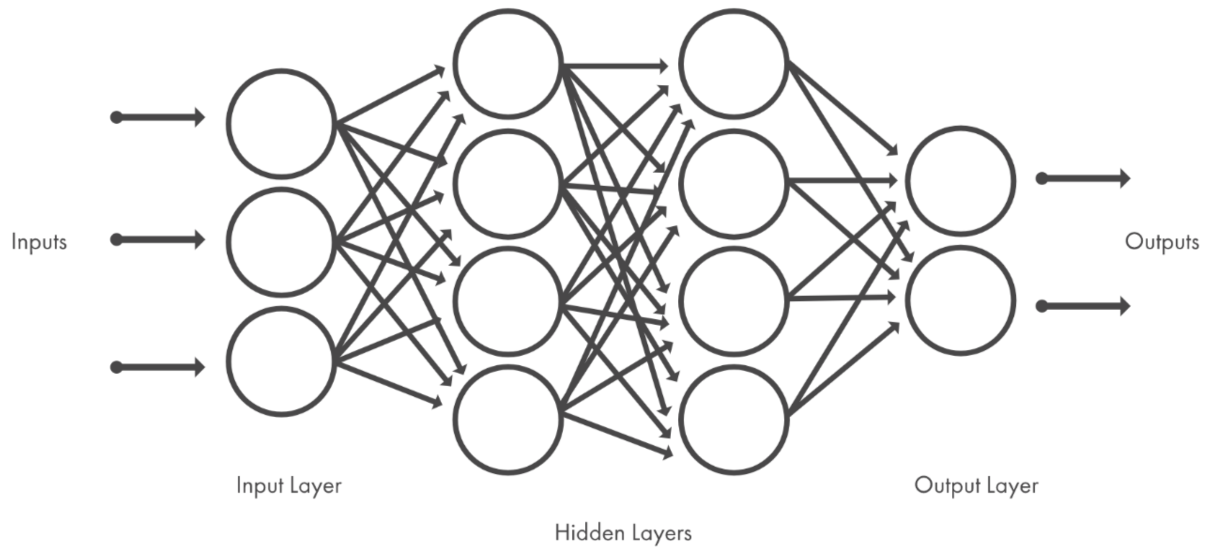


Figure 3.1: Neural networks, which are organized in layers consisting of a set of interconnected nodes. Networks can have tens or hundreds of hidden layers.

3.2 NEURAL NETWORK

3.2.1 Definition

Neural networks are a set of algorithms, modelled loosely after the human brain, that is designed to recognize patterns. They interpret sensory data through a kind of machine perception, labelling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of the data you store and manage. They help to group unlabelled data according to similarities among the example inputs, and they classify data when they have a labelled dataset to train on. (Neural networks can also extract features that are fed to other algorithms for clustering and classification; so you can think of deep neural networks as components of larger machine-learning applications involving algorithms for reinforcement learning, classification and regression.)

3.2.2 Classification

All classification tasks depend upon labelled datasets; that is, humans must transfer their knowledge to the dataset for a neural network to learn the correlation between labels and data. This is known as *supervised learning*.

- Detect faces, identify people in images, recognize facial expressions (angry, joyful)
- Identify objects in images (stop signs, pedestrians, lane markers…)
- Recognize gestures in video
- Detect voices, identify speakers, transcribe speech to text, recognize sentiment in voice.
- Classify text as spam (in emails), or fraudulent (in insurance claims); recognize sentiment in a text (customer feedback)

Any labels that humans can generate, any outcomes that you care about and which correlate to data, can be used to train a neural network.

3.2.3 Clustering

Clustering or grouping is the detection of similarities. Deep learning does not require labels to detect similarities. Learning without labels is called *unsupervised learning*. Unlabelled data is the majority of data in the world. One law of machine learning is: the more data an algorithm can train on, the more accurate it will be. Therefore, unsupervised learning has the potential to produce highly accurate models.

- Search: Comparing documents, images or sounds to surface similar items.
- Anomaly detection: The flipside of detecting similarities is detecting anomalies or unusual behaviour. In many cases, unusual behaviour correlates highly with things you want to detect and prevent, such as fraud.

3.2.4 Predictive Analytics: Regressions

With classification, deep learning can establish correlations between, say, pixels in an image and the name of a person. You might call this a static prediction. By the same token, exposed to enough of the right data, deep learning can establish correlations between present events and future events. It can run regression between the past and the future. The future event is like the label in a sense. Deep learning doesn't necessarily care about time or the fact that something hasn't happened yet. Given a time series, deep learning may read a string of number and predict the number most likely to occur next.

- Hardware breakdowns (data centres, manufacturing, transport)
- Health breakdowns (strokes, heart attacks based on vital stats and data from wearables)
- Customer churn (predicting the likelihood that a customer will leave, based on web activity and metadata)
- Employee turnover (ditto, but for employees)

The better we can predict, the better we can prevent and pre-empt. As you can see, with neural networks, we're moving towards a world of fewer surprises. Not zero surprises, just marginally fewer. We're also moving toward a world of smarter agents that combine neural networks with other algorithms like reinforcement learning to attain goals.

With that brief overview of deep learning use cases, let's look at what neural nets are made of.

Working of a Neural Networks

A neural network has a large number of processors. These processors operate parallelly but are arranged as tiers. The first tier receives the raw input similar to how the optic nerve receives the raw information in human beings.

Each successive tier then receives input from the tier before it and then passes on its output to the tier after it. The last tier processes the final output.

Small nodes make up each tier. The nodes are highly interconnected with the nodes in the tier before and after. Each node in the neural network has its own sphere of knowledge, including rules that it was programmed with and rules it has learnt by itself.

The key to the efficacy of neural networks is they are extremely adaptive and learn very quickly. Each node weighs the importance of the input it receives from the nodes before it. The inputs that contribute the most towards the right output are given the highest weight.

Different Types of Neural Networks

Different types of neural networks use different principles in determining their own rules. There are many types of artificial neural networks, each with their unique strengths. You can take a look at this video to see the different types of neural networks and their applications in detail.

3.2.5 Types of neural networks and their applications.

- **Feedforward Neural Network – Artificial Neuron**

This is one of the simplest types of artificial neural networks. In a feedforward neural network, the data passes through the different input nodes until it reaches the output node. In other words, data moves in only one direction from the first tier onwards until it reaches the output node. This is also known as a front propagated wave which is usually achieved by using a classifying activation function. Unlike in more complex types of neural networks, there is no backpropagation and data moves in one direction only. A feedforward neural network may have a single layer or it may have hidden layers.

In a feedforward neural network, the sum of the products of the inputs and their weights are calculated. This is then fed to the output. Here is an example of a single layer feedforward neural network.

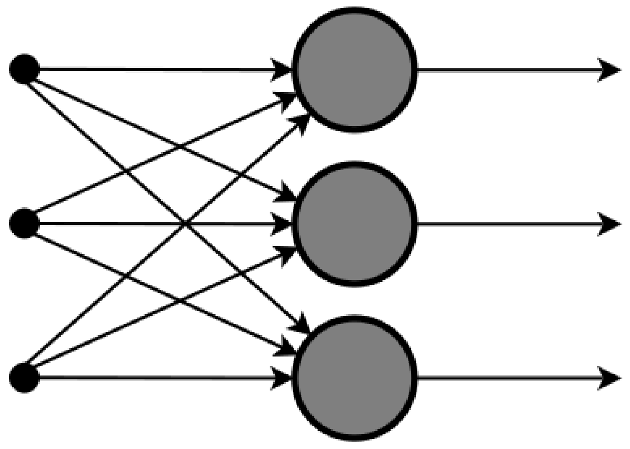


Figure 3.2 Feedforward Neural Network – Artificial Neuron

Feedforward neural networks are used in technologies like face recognition and computer vision. This is because the target classes in these applications are hard to classify.

A simple feedforward neural network is equipped to deal with data which contains a lot of noise. Feedforward neural networks are also relatively simple to maintain.

- **Radial Basis Function Neural Network**

A radial basis function considers the distance of any point relative to the centre. Such neural networks have two layers. In the inner layer, the features are combined with the radial basis function.

Then the output of these features is taken into account when calculating the same output in the next time-step. Here is a diagram which represents a radial basis function neural network.

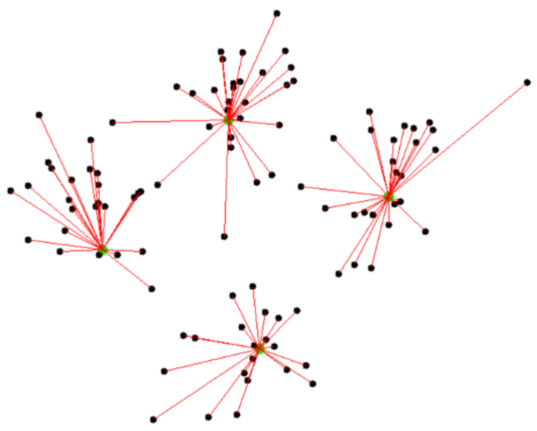


Figure 3.3 Radial Basis Function Neural Network

The radial basis function neural network is applied extensively in power restoration systems. In recent decades, power systems have become bigger and more complex.

This increases the risk of a blackout. This neural network is used in the power restoration systems to restore power in the shortest possible time.

- **Multilayer Perceptron**

A multilayer perceptron has three or more layers. It is used to classify data that cannot be separated linearly. It is a type of artificial neural network that is fully connected. This is because every single node in a layer is connected to each node in the following layer. A multilayer perceptron uses a nonlinear activation function (mainly hyperbolic tangent or logistic function). Here’s what a multilayer perceptron looks like.

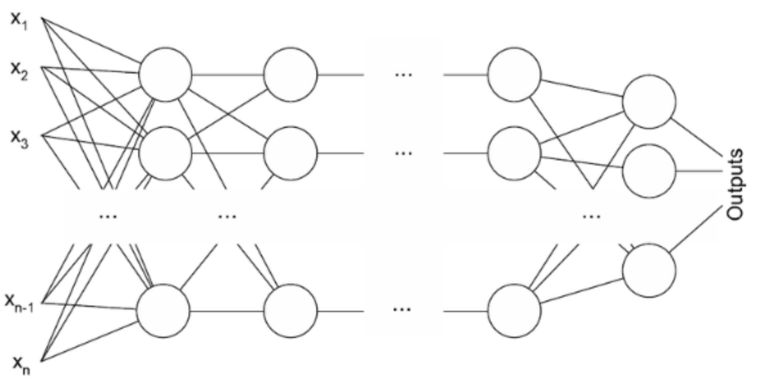


Figure 3.4 Multilayer Perceptron

This type of neural network is applied extensively in speech recognition and machine translation technologies.

- **Convolutional Neural Network**

A convolutional neural network (CNN) uses a variation of the multilayer perceptron. A CNN contains one or more than one convolutional layer. These layers can either be completely interconnected or pooled. Before passing the result to the next layer, the convolutional layer uses a convolutional operation on the input. Due to this convolutional operation, the network can be much deeper but with much fewer parameters. Due to this ability, convolutional neural networks show very effective results in image and video recognition, natural language processing, and recommender systems. Convolutional neural networks also show great results in semantic parsing and paraphrase detection. They are also applied in signal processing and image classification.

CNN's are also being used in image analysis and recognition in agriculture where weather features are extracted from satellites like LSAT to predict the growth and yield of a piece of land. Here's an image of what a Convolutional Neural Network looks like.

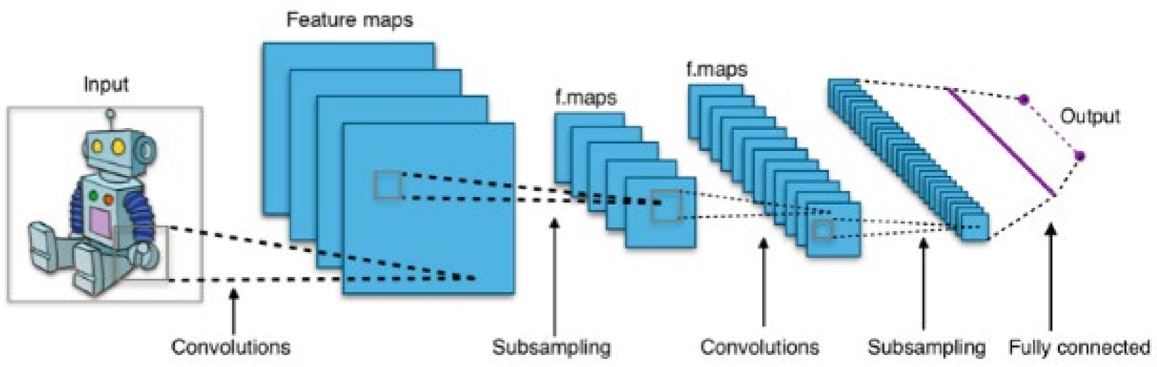


Figure 3.5Convolutional Neural Network

- **Recurrent Neural Network (RNN) – Long Short-Term Memory**

A Recurrent Neural Network is a type of artificial neural network in which the output of a particular layer is saved and fed back to the input. This helps predict the outcome of the layer.

The first layer is formed in the same way as it is in the feedforward network. That is, with the product of the sum of the weights and features. However, in subsequent layers, the recurrent neural network process begins.

From each time-step to the next, each node will remember some information that it had in the previous time-step. In other words, each node acts as a memory cell while computing and carrying out operations. The neural network begins with the front propagation as usual but remembers the information it may need to use later.

If the prediction is wrong, the system self-learns and works towards making the right prediction during the backpropagation. This type of neural network is very effective in text-to-speech conversion technology. Here’s what a recurrent neural network looks like.

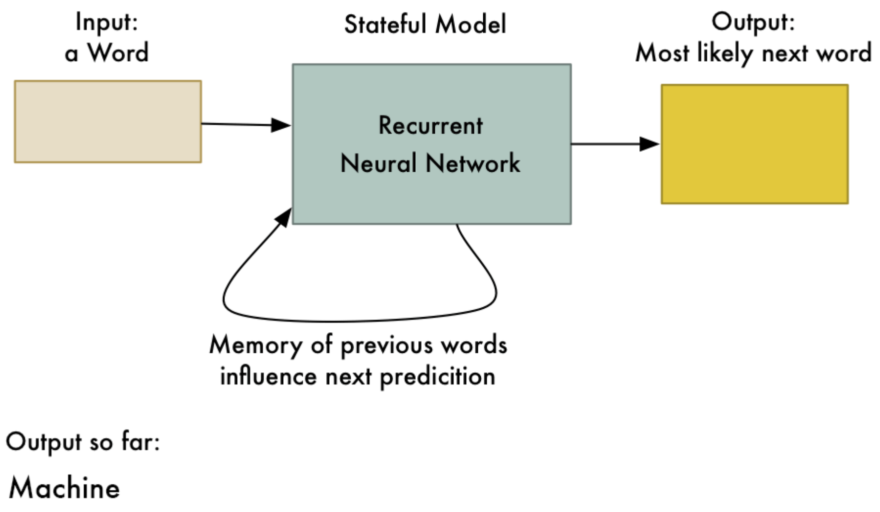


Figure 3.6 Recurrent Neural Network (RNN) – Long Short-Term Memory

- **Modular Neural Network**

A modular neural network has a number of different networks that function independently and perform sub-tasks. The different networks do not interact with or signal each other during the computation process. They work independently towards achieving the output.

As a result, a large and complex computational process can be done significantly faster by breaking it down into independent components. The computation speed increases because the networks are not interacting with or even connected. Here's a visual representation of a Modular Neural Network.

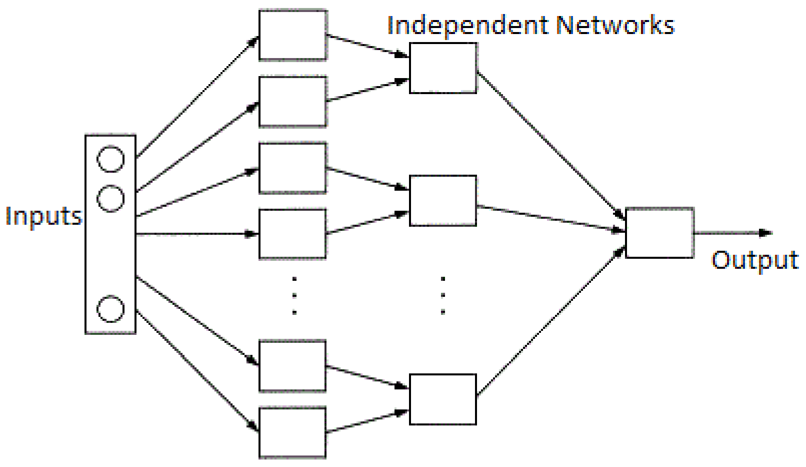


Figure 3.7Modular Neural Network

- **Sequence-To-Sequence Models**

A sequence to sequence model consists of two recurrent neural networks. There's an encoder that processes the input and a decoder that processes the output. The encoder and decoder can either use the same or different parameters. This model is particularly applicable in those cases where the length of the input data is not the same as the length of the output data.

Sequence-to-sequence models are applied mainly in chatbots, machine translation, and question answering systems.

3.3 RECURRENT NEURAL NETWORKS

Recurrent neural networks are feedforward neural networks augmented by the inclusion of edges that span adjacent time steps, introducing a notion of time to the model. Like feedforward networks, RNNs may not have cycles among conventional edges. However, edges that connect adjacent time steps, called recurrent edges, may form cycles, including cycles of length one that is self- connections from a node to itself across time. At time t, nodes with recurrent edges receive input from the current data point x(t) and also from hidden node values h(t−1) in the network's previous state. The output yˆ(t) at each time t is calculated given the hidden node values h(t) at time t. Input x(t−1) at time t 1 can influence the output yˆ(t) at time t and later by way of the recurrent connections. Two equations specify all calculations necessary for computation at each time step on the forward pass in a simple recurrent neural network as in Figure 3.8:

$$\boldsymbol{h}^{(t)} = \sigma\,(W^{\mathrm{hx}}\boldsymbol{x}^{(t)} + W^{\mathrm{hh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$

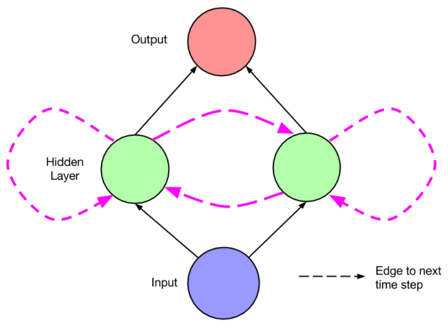


Figure 3.8 A simple recurrent network. At each time step t, activation is passed along solid edges as in a feedforward network. Dashed edges connect a source node at each time t to a target node at each following time t + 1.

$$\hat{\boldsymbol{y}}^{(t)}=\mathrm{softmax}(W^{\mathrm{yh}}\boldsymbol{h}^{(t)}+\boldsymbol{b}_y).$$

Here W^{hx} is the matrix of conventional weights between the input and the hidden layer and W^{hh} is the matrix of recurrent weights between the hidden layer and itself at adjacent time steps. The vectors b_h and b_y are bias parameters which allow each node to learn an offset. The dynamics of the network depicted in Figure 3 across time steps can be visualized by unfolding it as in Figure 4. Given this picture, the network can be interpreted not as cyclic, but rather as a deep network with one layer per time step and shared weights across time steps. It is then clear that the unfolded network can be trained across many time steps using backpropagation. This algorithm, called backpropagation through time (BPTT), was introduced by Werbos [1990]. All recurrent networks in common current use apply it.

The most successful RNN architectures for sequence learning stem from two papers published in 1997. The first paper, Long Short-Term Memory by Hochreiter and Schmidhuber [1997], introduces the memory cell, a unit of computation that replaces traditional nodes in the hidden layer of a network. With these memory cells, networks can overcome difficulties with training encountered by earlier recurrent networks. The second paper, Bidirectional Recurrent Neural Networks by Schuster and Paliwal [1997], introduces an architecture in which information from both the future and the past are used to determine the output at any point in the sequence. This is in contrast to previous networks, in which only past input can affect the output, and has been used successfully for sequence labelling tasks in natural language processing, among others. Fortunately, the two innovations are not mutually exclusive and have been successfully combined for phoneme classification [Graves and Schmidhuber, 2005] and handwriting recognition [Graves et al., 2009]. In this section, we explain the LSTM and BRNN and we describe the neural Turing machine (NTM), which extends RNNs with an addressable external memory [Graves et al., 2014].

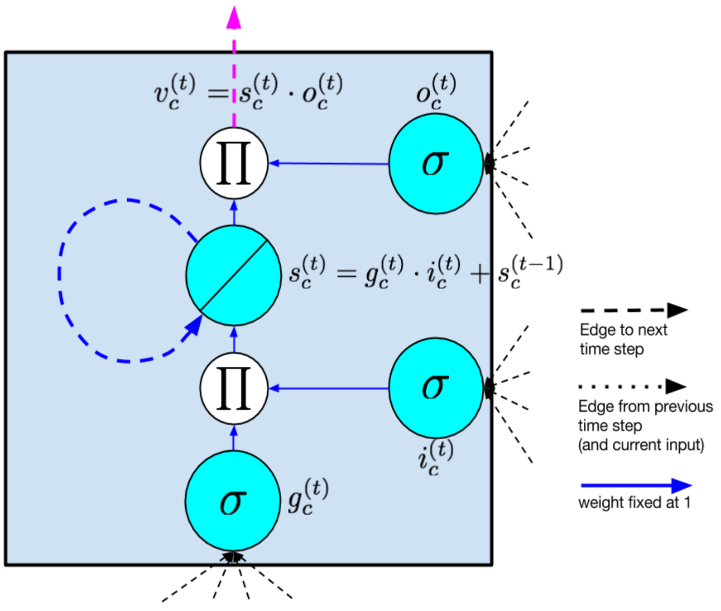

$v_c^{(t)} = s_c^{(t)} \cdot o_c^{(t)}$
$o_c^{(t)}$
$s_c^{(t)} = g_c^{(t)} \cdot i_c^{(t)} + s_c^{(t-1)}$
$i_c^{(t)}$
$g_c^{(t)}$
Edge to next time step
Edge from previous time step (and current input)
weight fixed at 1

Figure 3.9 One LSTM memory cell as proposed by Hochreiter and Schmidhuber [1997].

The self-connected node is the internal states. The diagonal line indicates that it is linear, i.e. the identity link function is applied. The blue dashed line is the recurrent edge, which has fixed unit weight. Nodes marked Π output the product of their inputs. All edges into and from Π nodes also have fixed unit weight.

3.3.1 Long short-term memory (LSTM)

Hochreiter and Schmidhuber [1997] introduced the LSTM model primarily in order to overcome the problem of vanishing gradients. This model resembles a standard recurrent neural network with a hidden layer, but each ordinary node in the hidden layer is replaced by a memory cell. Each memory cell contains a node with a self-connected recurrent edge of fixed weight one, ensuring that the gradient can pass across many time steps without vanishing or exploding. To distinguish references to a memory cell and not an ordinary node, we use the subscript c. The term "long short-term memory" comes from the following intuition. Simple recurrent neural networks have long-term memory in the form of weights. The weights change slowly during training, encoding general knowledge about the data. They also have a short-term memory in the form of ephemeral activations, which pass from each node to successive nodes. The LSTM model introduces an intermediate type of storage via the memory cell. A memory cell is a composite unit, built from simpler nodes in a specific connectivity pattern, with the novel inclusion of multiplicative nodes, represented in diagrams by the letter Π. All elements of the LSTM cell are enumerated and described below. Note that when we use vector notation, we are referring to the values of the nodes in an entire layer of cells. For example, s is a vector containing the value of s_c at each memory cell c in a layer. When the subscript c is used, it is to index an individual memory cell.

- Input node: This unit, labelled g_c, is a node that takes activation in the standard way from the input layer x(t) at the current time step and (along recurrent edges) from the hidden layer at the previous time step h(t−1). Typically, the summed weighted input is run through a tanh activation function, although in the original LSTM paper, the activation function is a sigmoid.

- Input gate: Gates is a distinctive feature of the LSTM approach. A gate is a sigmoidal unit that, like the input node, takes activation from the current data point x(t) as well as from the hidden layer at the previous time step. A gate is so-called because its value is used to multiply the value of another node. It is a gate in the sense that if its value is zero, then flow from the other node is cut off. If the value of the gate is one, all flow is passed through. The value of the input gate i_c multiplies the value of the input node.

- Internal state: At the heart of each memory cell is a node s_c with linear activation, which is referred to in the original paper as the "internal state" of the cell. The internal state s_c has a self-connected recurrent edge with fixed unit weight. Because this edge spans adjacent time steps with constant weight, an error can flow across time steps without vanishing or exploding. This edge is often called the constant error carousel. In vector notation, the update for the internal state is $s(t) = g(t) \, 0 \, i(t) + s(t-1)$ where 0 is pointwise multiplication.

- Forget gate: These gates f_c was introduced by Gers et al. [2000]. They provide a method by which the network can learn to flush the contents of the internal state. This is especially useful in continuously running networks. With forget gates, the equation to calculate the internal state on the forward pass is $\mathbf{s(t)} = g^{(t)} \, 0 \, i^{(i)} + s^{(t-1)} \, 0 \, f^{(t)}$

- Output gate: The value v_c ultimately produced by a memory cell is the value of the internal state s_c multiplied by the value of the output gate oc. It is customary that the internal state first is run through a tanh activation function, as this gives the output of each cell the same dynamic range as an ordinary tanh hidden unit. However, in other neural network research, rectified linear units, which have a greater dynamic range, are easier to train. Thus, it seems plausible that the nonlinear function on the internal state might be omitted.

In the original paper and most subsequent work, the input node is labelled g. We adhere to this convention but note that it may be confusing as g does not stand for gate. In the original paper, the gates are called yin and y_{out} but this is confusing because y generally stands for output in the machine learning literature. Seeking comprehensibility, we break with this convention and use i, f , and o to refer to input, forget and output gates respectively, as in Sutskever et al. [2014].

Since the original LSTM was introduced, several variations have been proposed. Forget gates, described above, were proposed in 2000 and were not part of the original LSTM design. However, they have proven effective and are standard in most modern implementations. That same year, Gers and Schmidhuber [2000] proposed peephole connections that pass from the internal state directly to the input and output gates of that same node without first having to be modulated by the output gate. They report that these connections improve performance on timing tasks where the network must learn to measure precise intervals between events. The intuition of the peephole connection can be captured by the following example. Consider a network which must learn to count objects and emit some desired output when n objects have been seen. The network might learn to let some fixed amount of activation into the internal state after each object is seen. This activation is trapped in the internal state sc by the constant error carousel and is incremented iteratively each time another object is seen. When the nth object is seen, the network needs to know to let out content from the internal state so that it can affect the output. To accomplish this, the output gate oc must know the content of the internal state sc. Thus sc should be an input to oc.

Put formally, computation in the LSTM model proceeds according to the following calculations, which are performed at each time step. These equations give the full algorithm for a modern LSTM with forget gates:

$$\mathbf{g(t)} = \varphi\,(W^{gx}\,x(t) + W^{gh}\,h^{(t-1)} + b_g\,)$$

$$\mathbf{i(t)} = \sigma(W^{ix}\,x(t) + W^{ih}\,h^{(t-1)} + b_i)$$

$$\mathbf{f\,(t)} = \sigma(W^{fx}\,x(t) + W^{fh}\,h^{(t-1)} + b_f\,)$$

$$\mathbf{o(t)} = \sigma(W^{ox}\, x(t) + W^{ho}\, h^{(t-1)} + b_o)$$

$$\mathbf{s(t)} = g^{(t)}\ 0\ i^{(i)} + s^{(t-1)}\ 0\ f^{(t)}$$

$$\mathbf{h(t)} = \varphi(s^{(t)})\ 0\ o^{(t)}$$

The value of the hidden layer of the LSTM at time t is the vector h(t), while h(t−1) is the values output by each memory cell in the hidden layer at the previous time. Note that these equations include the forget gate, but not peephole connections. The calculations for the simpler LSTM without forget gates are obtained by setting f (t) = 1 for all t. We use the tanh function φ for the input node g following the state-of-the-art design of Zaremba and Sutskever [2014]. However, in the original LSTM paper, the activation function for g is the sigmoid σ.

While deep neural networks are all the rage, the complexity of the major frameworks has been a barrier to their use for developers new to machine learning. There have been several proposals for improved and simplified high-level APIs for building neural network models, all of which tend to look similar from a distance but show differences on closer examination.

3.4 LIBRARIES

3.4.1 Keras

Keras is one of the leading high-level neural networks APIs. It is written in Python and supports multiple back-end neural network computation engines. Keras is an open-source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the exception deep neural network model.

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than a standalone machine learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Microsoft added a CNTK backend to Keras as well, available as of CNTK v2.0.

Keras proper does not do its low-level operations, such as tensor products and convolutions; it relies on a back-end engine for that. Even though Keras supports multiple back-end engines, its primary (and default) back end is TensorFlow, and its primary supporter is Google. The Keras API comes packaged in TensorFlow as tf.keras, which as mentioned earlier will become the primary TensorFlow API as of TensorFlow 2.0.

Keras models

The Model is the core Keras data structure. There are two main types of models available in Keras: The **Sequential model**, and the Model class used with the **functional API**.

- Keras Sequential models

The Sequential model is a linear stack of layers, and the layers can be described very simply. Sequential groups a linear stack of layers into a tf.keras. Model. Sequential provides training and inference features on this model.

- Keras functional API

The Keras Sequential model is simple but limited in model topology. The Keras functional API is useful for creating complex models, such as multi-input/multi-output models, directed acyclic graphs (DAGs), and models with shared layers.

The functional API uses the same layers as the Sequential model but provides more flexibility in putting them together. In the functional API you define the layers first, and then create the Model, compile it, and fit (train) it. Evaluation and prediction are essentially the same as in a Sequential model.

Keras layers

Keras layers are the primary building block of Keras models. Each layer receives input information, do some computation and finally output the transformed information. The output of one layer will flow into the next layer as its input.

A Keras layer requires the shape of the input (input_shape) to understand the structure of the input data, initializer to set the weight for each input and finally activators to transform the output to make it non-linear. In between, constraints restrict and specify the range in which the weight of input data to be generated and regularizer will try to optimize the layer (and the model) by dynamically applying the penalties on the weights during the optimization process.

Keras layer requires below minimum details to create a complete layer.

- The shape of the input data
- Number of neurons/units in the layer
- Initializers
- Regularizers
- Constraints
- Activations

3.4.2 Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

3.4.3 NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

3.4.4Scikit-learn

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy

3.4.5Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

Matplotlib was originally written by John D. Hunter, since then it has an active development community, and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012, and further joined by Thomas Caswell.

CHAPTER 4 - IMPLEMENTATION

4.1 PREDICTIVE MAINTENANCE USING LONG SHORT-TERM MEMORY

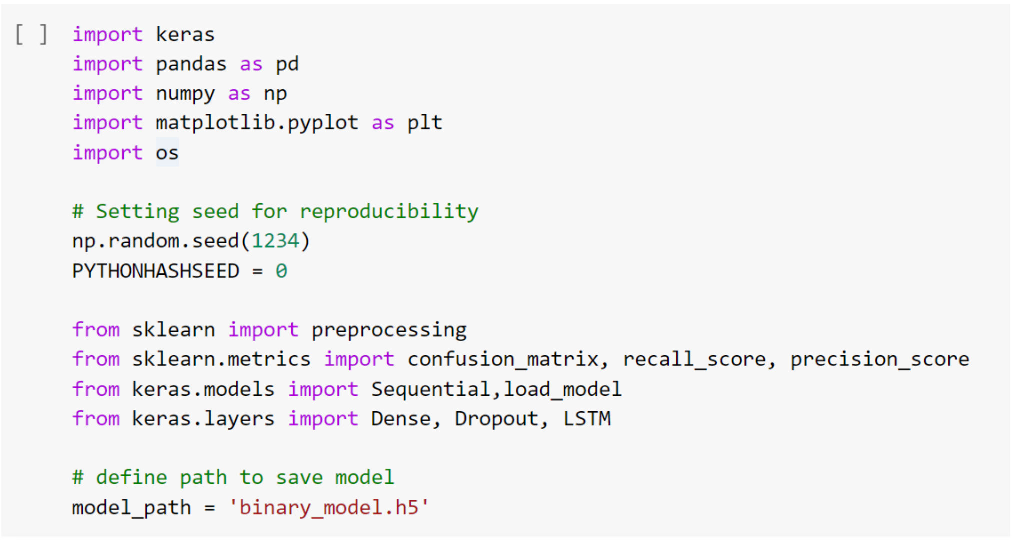
For demonstrating predictive maintenance, we'll use the simulated data provided in NASA Repository The dataset consists of the following three files:

- Training data: It contains the aircraft engine run to failure data
- Testing data: It contains the aircraft engine operating data without failure events recorded.
- Ground truth data: Here, the information about the true remaining cycles for each engine in testing data is available.

According to the data description provided at the data source, the training data consists of multiple multivariate time series with the cycle as the time unit, together with 21 sensors reacting for each cycle. Each time series can be assumed as being generated from a different engine of the same type. Each engine is assumed to start with different degrees of initial wear and manufacturing variation, and this information is unknown to the user. In this simulated data, the engine is assumed to be operating normally at the start of each time series. It starts to degrade at some point during the series of the operating cycles. This degrades the progress and grows in magnitude. When a predefined threshold is reached, then the engine is considered unsafe for further operation. In other words, the last cycle in each time series can be considered as the failure point of the corresponding engine. Taking the sample training data as an example, the engine with id=1 fails at cycle 192, and engine with id=2 fails at cycle 287.
The testing data has the same data schema as the training data. The only difference is that the data does not indicate when the failure occurs (in other words, the last time period does NOT represent the failure point). Taking the sample testing data, the engine with id=l runs from cycle 1 through cycle 31. It is not shown how many more cycles this engine can last before it fails.
The ground truth data provides the number of remaining working cycles £or the engines in the testing data. Taking the sample ground truth data shown as an example, the engine with id=l in the testing data can run another 112 cycles before it fails.

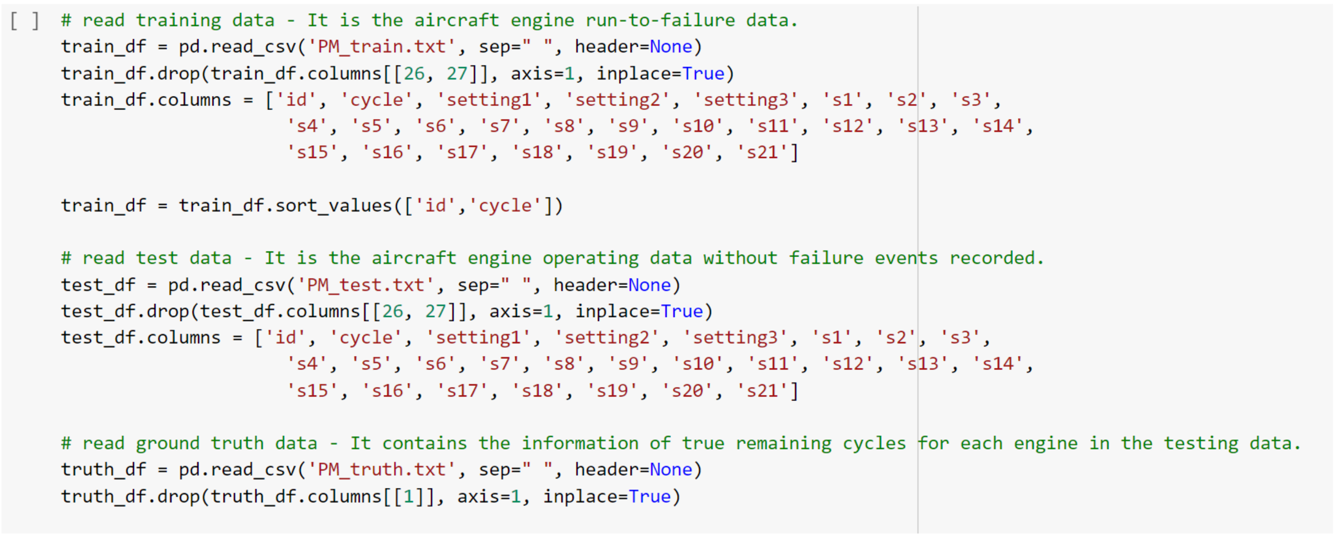
Since this is a time-series data, we will use Long Short-Term Memory (LSTM) to classify whether the engine will fail in a certain time period or not.

1. The modules needed to implement predictive maintenance are imported in the first step. We also set the seed for random calculations so that the result is reproducible:



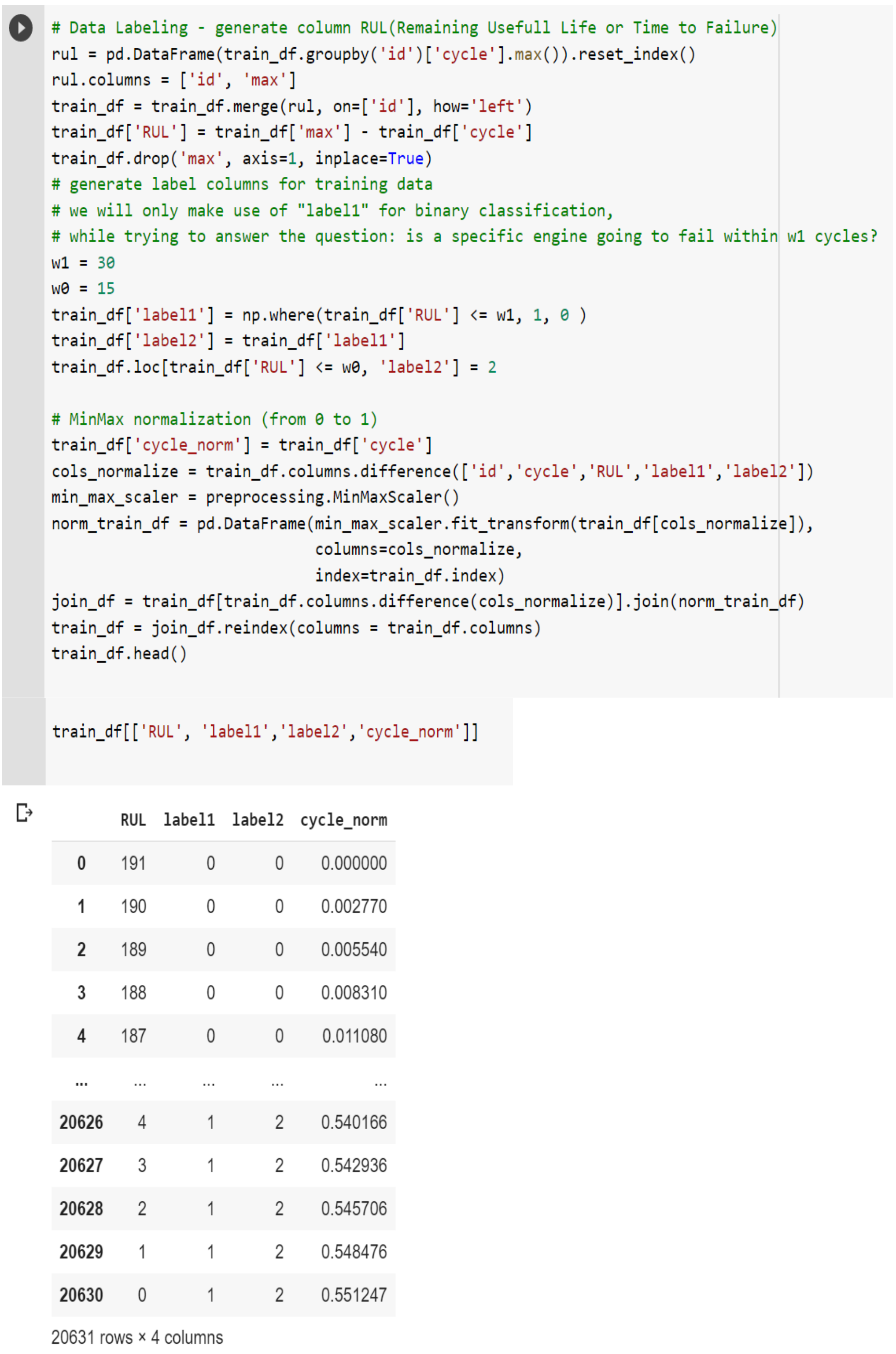
```
[ ]  import keras
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import os

     # Setting seed for reproducibility
     np.random.seed(1234)
     PYTHONHASHSEED = 0

     from sklearn import preprocessing
     from sklearn.metrics import confusion_matrix, recall_score, precision_score
     from keras.models import Sequential,load_model
     from keras.layers import Dense, Dropout, LSTM

     # define path to save model
     model_path = 'binary_model.h5'
```

2. Let's read the data and assign column names, shown in the following code:



```
[ ]  # read training data - It is the aircraft engine run-to-failure data.
     train_df = pd.read_csv('PM_train.txt', sep=" ", header=None)
     train_df.drop(train_df.columns[[26, 27]], axis=1, inplace=True)
     train_df.columns = ['id', 'cycle', 'setting1', 'setting2', 'setting3', 's1', 's2', 's3',
                         's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14',
                         's15', 's16', 's17', 's18', 's19', 's20', 's21']

     train_df = train_df.sort_values(['id','cycle'])

     # read test data - It is the aircraft engine operating data without failure events recorded.
     test_df = pd.read_csv('PM_test.txt', sep=" ", header=None)
     test_df.drop(test_df.columns[[26, 27]], axis=1, inplace=True)
     test_df.columns = ['id', 'cycle', 'setting1', 'setting2', 'setting3', 's1', 's2', 's3',
                         's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14',
                         's15', 's16', 's17', 's18', 's19', 's20', 's21']

     # read ground truth data - It contains the information of true remaining cycles for each engine in the testing data.
     truth_df = pd.read_csv('PM_truth.txt', sep=" ", header=None)
     truth_df.drop(truth_df.columns[[1]], axis=1, inplace=True)
```

3. As the first step, we make a prediction whether the engine will fail in the time period or not, hence our label will be 1 or 0, that is, this will be a binary classification problem. To create the· binary Label, we preprocess the data and we create a new label remaining useful life (RUL). We also create a binary label1 variable telling if a specific engine is going to fail within wl cycles or not. And finally, the data (non-sensor) is normalized, shown as follows:

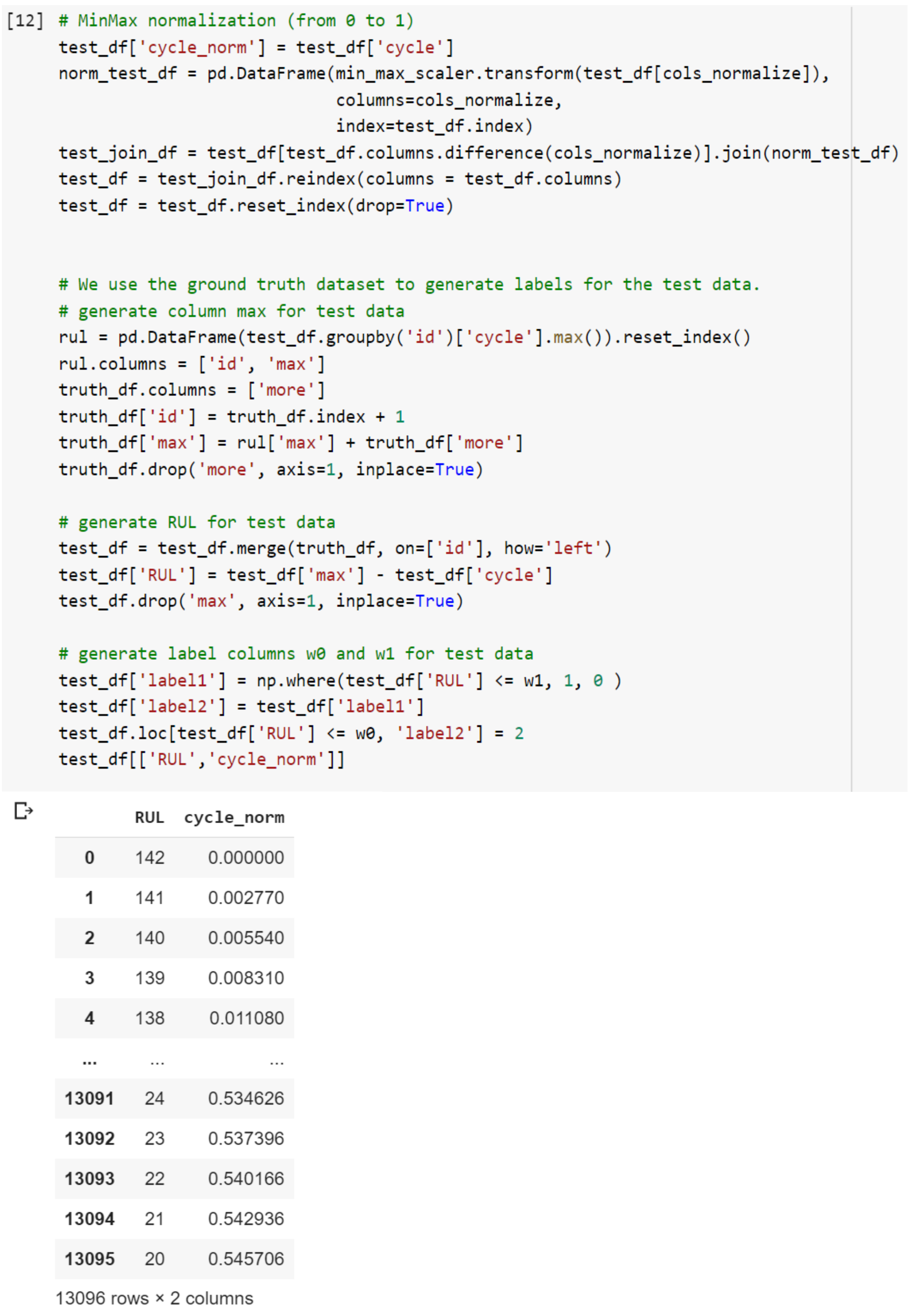


```python
# Data Labeling - generate column RUL(Remaining Usefull Life or Time to Failure)
rul = pd.DataFrame(train_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
train_df = train_df.merge(rul, on=['id'], how='left')
train_df['RUL'] = train_df['max'] - train_df['cycle']
train_df.drop('max', axis=1, inplace=True)
# generate label columns for training data
# we will only make use of "label1" for binary classification,
# while trying to answer the question: is a specific engine going to fail within w1 cycles?
w1 = 30
w0 = 15
train_df['label1'] = np.where(train_df['RUL'] <= w1, 1, 0 )
train_df['label2'] = train_df['label1']
train_df.loc[train_df['RUL'] <= w0, 'label2'] = 2

# MinMax normalization (from 0 to 1)
train_df['cycle_norm'] = train_df['cycle']
cols_normalize = train_df.columns.difference(['id','cycle','RUL','label1','label2'])
min_max_scaler = preprocessing.MinMaxScaler()
norm_train_df = pd.DataFrame(min_max_scaler.fit_transform(train_df[cols_normalize]),
                             columns=cols_normalize,
                             index=train_df.index)
join_df = train_df[train_df.columns.difference(cols_normalize)].join(norm_train_df)
train_df = join_df.reindex(columns = train_df.columns)
train_df.head()
```

```python
train_df[['RUL', 'label1','label2','cycle_norm']]
```

```
        RUL  label1  label2  cycle_norm
    0   191       0       0    0.000000
    1   190       0       0    0.002770
    2   189       0       0    0.005540
    3   188       0       0    0.008310
    4   187       0       0    0.011080
  ...   ...     ...     ...         ...
20626     4       1       2    0.540166
20627     3       1       2    0.542936
20628     2       1       2    0.545706
20629     1       1       2    0.548476
20630     0       1       2    0.551247

20631 rows × 4 columns
```

4. Similar preprocessing is performed on the test dataset, with just one change-theRUL value is obtained from the ground truth data:

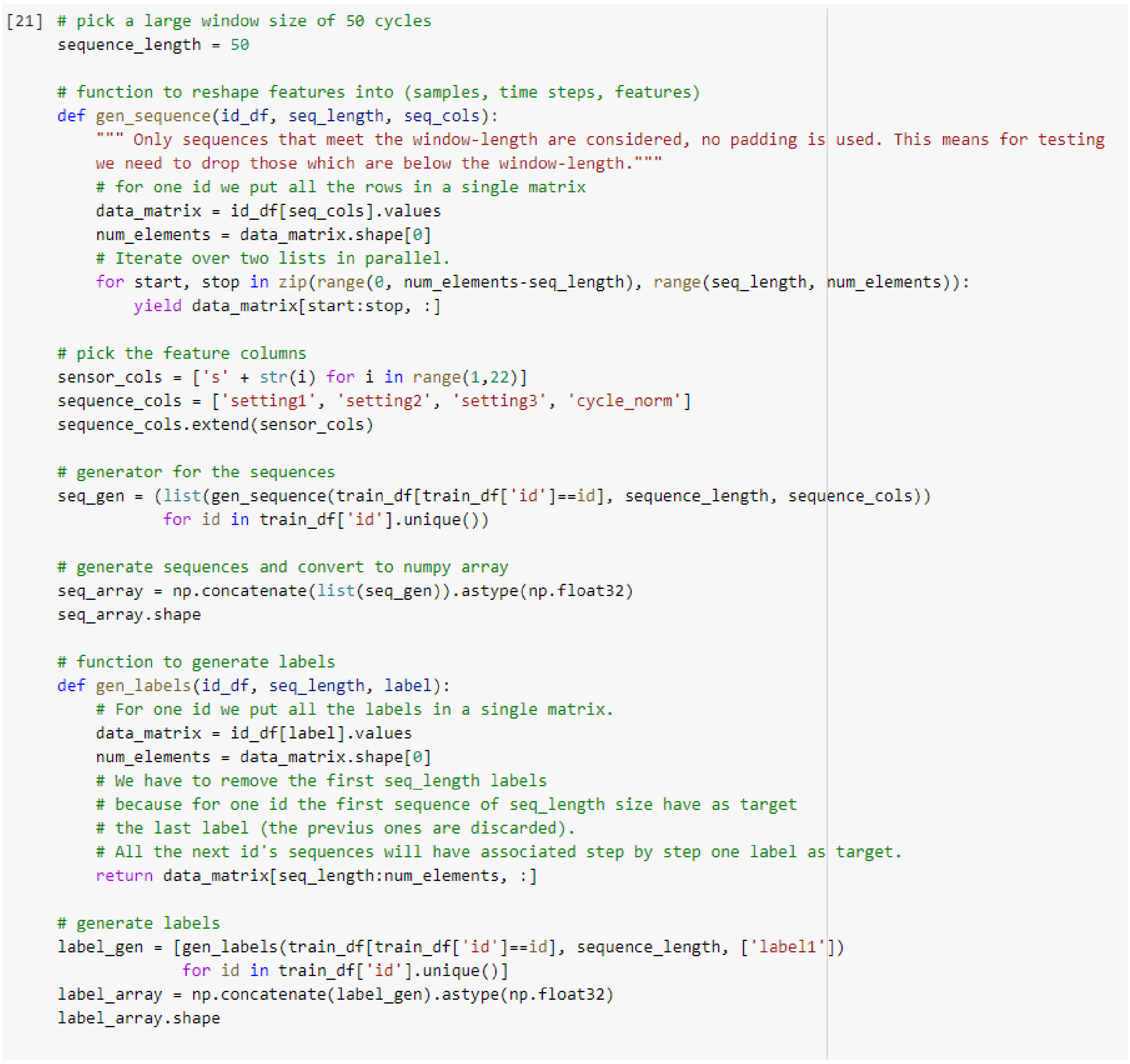


```
[12] # MinMax normalization (from 0 to 1)
     test_df['cycle_norm'] = test_df['cycle']
     norm_test_df = pd.DataFrame(min_max_scaler.transform(test_df[cols_normalize]),
                                 columns=cols_normalize,
                                 index=test_df.index)
     test_join_df = test_df[test_df.columns.difference(cols_normalize)].join(norm_test_df)
     test_df = test_join_df.reindex(columns = test_df.columns)
     test_df = test_df.reset_index(drop=True)


     # We use the ground truth dataset to generate labels for the test data.
     # generate column max for test data
     rul = pd.DataFrame(test_df.groupby('id')['cycle'].max()).reset_index()
     rul.columns = ['id', 'max']
     truth_df.columns = ['more']
     truth_df['id'] = truth_df.index + 1
     truth_df['max'] = rul['max'] + truth_df['more']
     truth_df.drop('more', axis=1, inplace=True)

     # generate RUL for test data
     test_df = test_df.merge(truth_df, on=['id'], how='left')
     test_df['RUL'] = test_df['max'] - test_df['cycle']
     test_df.drop('max', axis=1, inplace=True)

     # generate label columns w0 and w1 for test data
     test_df['label1'] = np.where(test_df['RUL'] <= w1, 1, 0 )
     test_df['label2'] = test_df['label1']
     test_df.loc[test_df['RUL'] <= w0, 'label2'] = 2
     test_df[['RUL','cycle_norm']]
```

	RUL	cycle_norm
0	142	0.000000
1	141	0.002770
2	140	0.005540
3	139	0.008310
4	138	0.011080
...	...	...
13091	24	0.534626
13092	23	0.537396
13093	22	0.540166
13094	21	0.542936
13095	20	0.545706

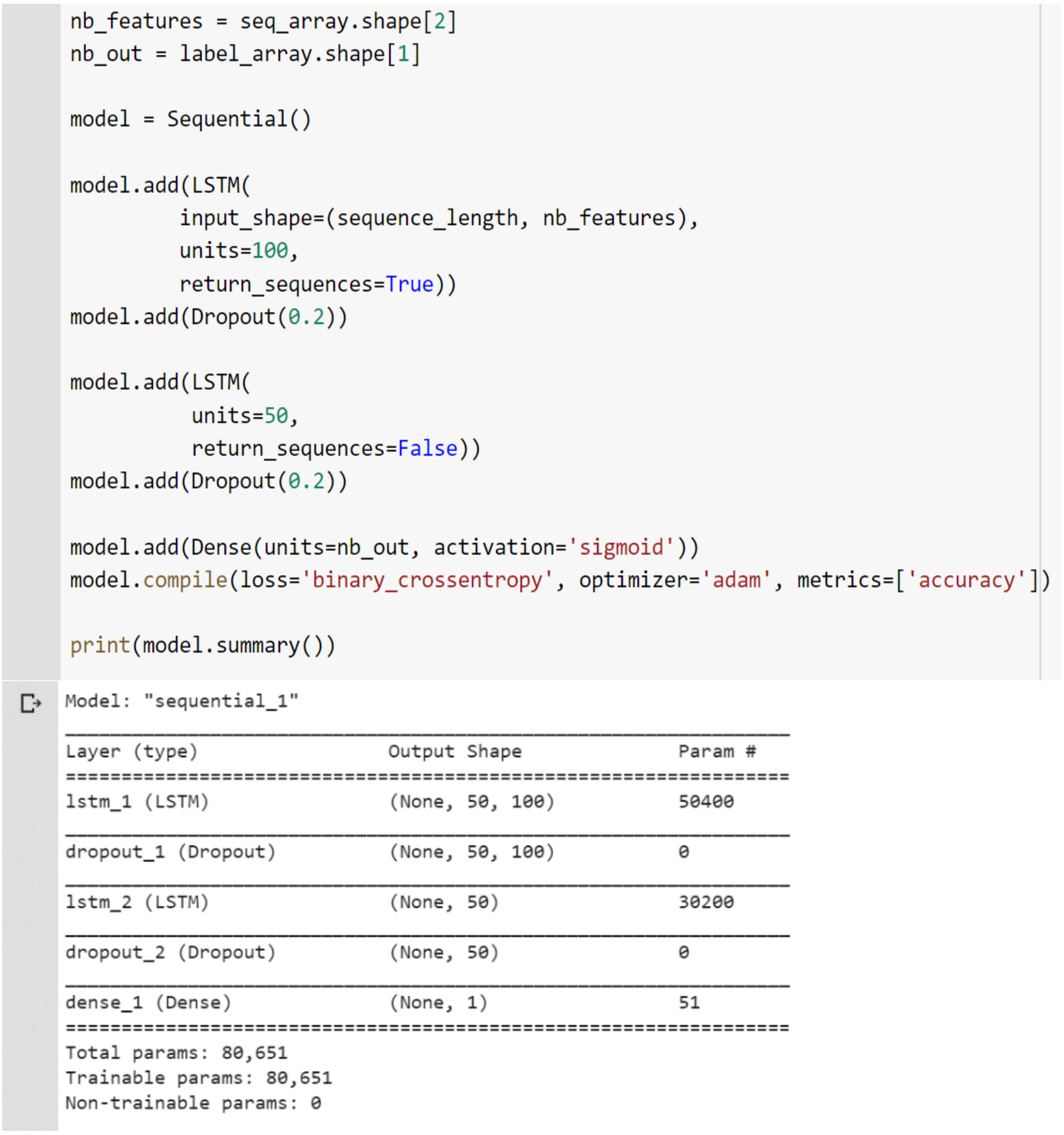
13096 rows × 2 columns

5. Since we are using LSTM for time-series modelling, we create a function that willgenerate the sequence to be fed to the LSTM as per the window size. We have chosen the window size of 50. We will also need a function to generate the corresponding label:



```
[21] # pick a large window size of 50 cycles
     sequence_length = 50

     # function to reshape features into (samples, time steps, features)
     def gen_sequence(id_df, seq_length, seq_cols):
         """ Only sequences that meet the window-length are considered, no padding is used. This means for testing
         we need to drop those which are below the window-length."""
         # for one id we put all the rows in a single matrix
         data_matrix = id_df[seq_cols].values
         num_elements = data_matrix.shape[0]
         # Iterate over two lists in parallel.
         for start, stop in zip(range(0, num_elements-seq_length), range(seq_length, num_elements)):
             yield data_matrix[start:stop, :]

     # pick the feature columns
     sensor_cols = ['s' + str(i) for i in range(1,22)]
     sequence_cols = ['setting1', 'setting2', 'setting3', 'cycle_norm']
     sequence_cols.extend(sensor_cols)

     # generator for the sequences
     seq_gen = (list(gen_sequence(train_df[train_df['id']==id], sequence_length, sequence_cols))
                for id in train_df['id'].unique())

     # generate sequences and convert to numpy array
     seq_array = np.concatenate(list(seq_gen)).astype(np.float32)
     seq_array.shape

     # function to generate labels
     def gen_labels(id_df, seq_length, label):
         # For one id we put all the labels in a single matrix.
         data_matrix = id_df[label].values
         num_elements = data_matrix.shape[0]
         # We have to remove the first seq_length labels
         # because for one id the first sequence of seq_length size have as target
         # the last label (the previus ones are discarded).
         # All the next id's sequences will have associated step by step one label as target.
         return data_matrix[seq_length:num_elements, :]

     # generate labels
     label_gen = [gen_labels(train_df[train_df['id']==id], sequence_length, ['label1'])
                  for id in train_df['id'].unique()]
     label_array = np.concatenate(label_gen).astype(np.float32)
     label_array.shape
```

6. Next, we build a deep network. The first layer is an LSTM layer with 100 units followed by another LSTM layer with 50 units. Dropout is also applied after each LSTM layer to control overfitting. The final layer is a Dense output layer with a single unit and sigmoid activation since this is a binary classification problem. We build the network.



```python
nb_features = seq_array.shape[2]
nb_out = label_array.shape[1]

model = Sequential()

model.add(LSTM(
         input_shape=(sequence_length, nb_features),
         units=100,
         return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(
          units=50,
          return_sequences=False))
model.add(Dropout(0.2))

model.add(Dense(units=nb_out, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 50, 100)           50400
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 100)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                30200
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 80,651
Trainable params: 80,651
Non-trainable params: 0
```

Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

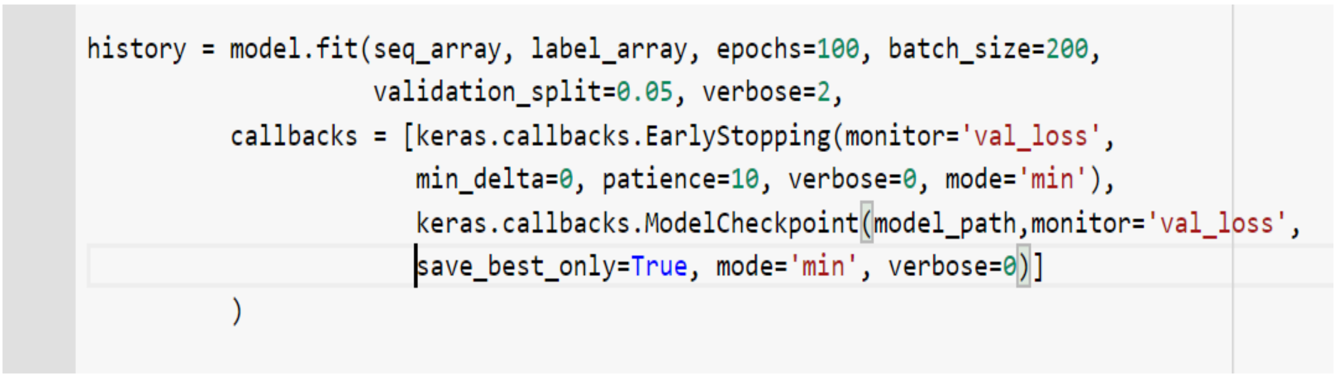
The loss function binary cross-entropy is used on yes/no decisions, e.g., multi-label classification. The loss tells you how wrong your model's predictions are.

Stacking LSTM hidden layers makes the model deeper, more accurately earning the description as a deep learning technique. It is the depth of neural networks that are generally attributed to the success of the approach on a wide range of challenging prediction problems.

Additional hidden layers can be added to a Multilayer Perceptron neural network to make it deeper. The additional hidden layers are understood to recombine the learned representation from prior layers and create new representations at high levels of abstraction. For example, from lines to shapes to objects.

A sufficiently large single hidden layer Multilayer Perceptron can be used to approximate most functions. Increasing the depth of the network provides an alternate solution that requires fewer neurons and trains faster. Ultimately, adding depth it is a type of representational optimization.

7. We train the model, shown as follows:



```
history = model.fit(seq_array, label_array, epochs=100, batch_size=200,
                    validation_split=0.05, verbose=2,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss',
                       min_delta=0, patience=10, verbose=0, mode='min'),
                       keras.callbacks.ModelCheckpoint(model_path,monitor='val_loss',
                       save_best_only=True, mode='min', verbose=0)]
          )
```

8. The trained model gives 98% accuracy on the test dataset and 97.8% accuracy onthe validation dataset. The precision value is 0.96, and there is a recall of 0.96 and an Fl score of 0.96. Not bad, right! The following diagram shows these results of the trained model:

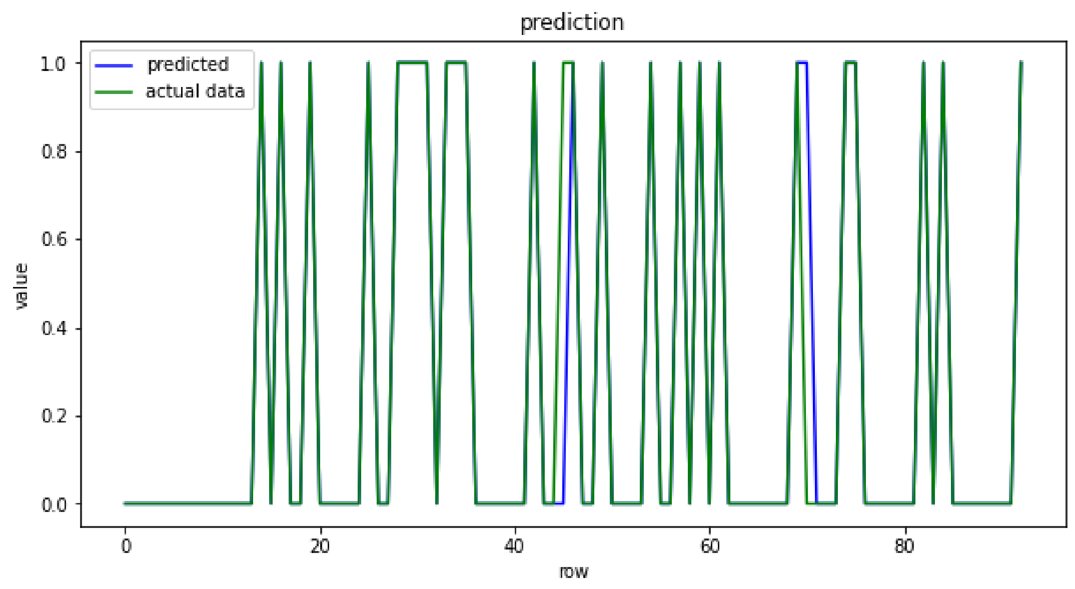
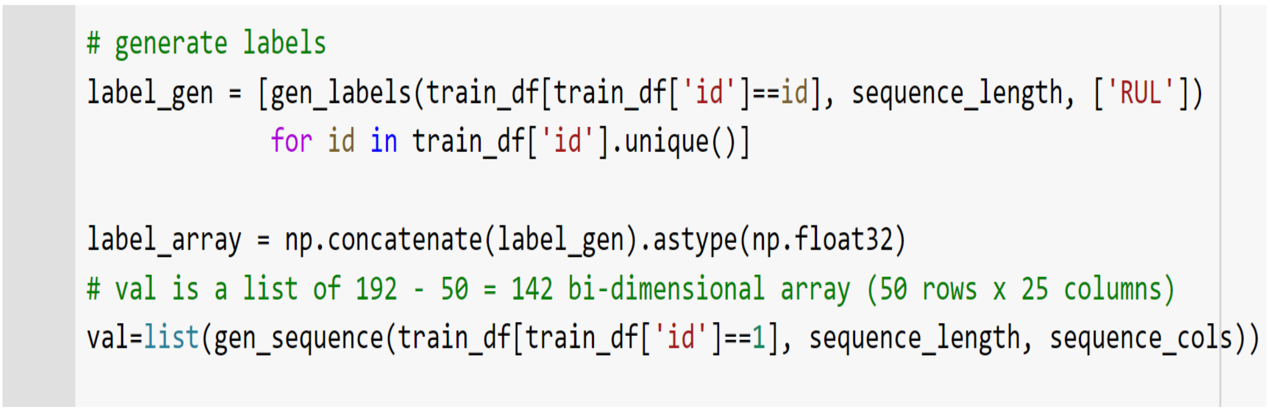


Figure 4.1 Result of the training model

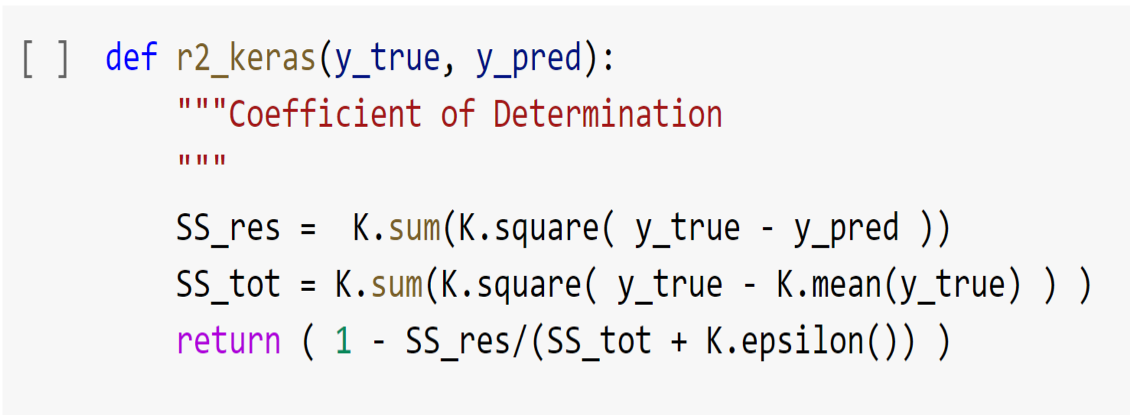
We can use the same data to also predict the RUL of the aircraft engines, that is, predict the engines time to failure. This will be a regression problem now we can use the LSTM model to perform regression as well. The initial steps will be the same as before, but from the fifth step onwards we will have changes. While the input data sequence generated will remain the same as before, the target will no longer be the binary label, instead, we will use RUL as the target for our regression model:

1. We create the target value using the same gen_labels () function. We also create a validation set using the gen_sequence () function:



```
# generate labels
label_gen = [gen_labels(train_df[train_df['id']==id], sequence_length, ['RUL'])
             for id in train_df['id'].unique()]

label_array = np.concatenate(label_gen).astype(np.float32)
# val is a list of 192 - 50 = 142 bi-dimensional array (50 rows x 25 columns)
val=list(gen_sequence(train_df[train_df['id']==1], sequence_length, sequence_cols))
```

2. Create an LSTM model. We are using r2 as the metrics during training, therefore, we use the Keras custom metric feature and our own metrics function:



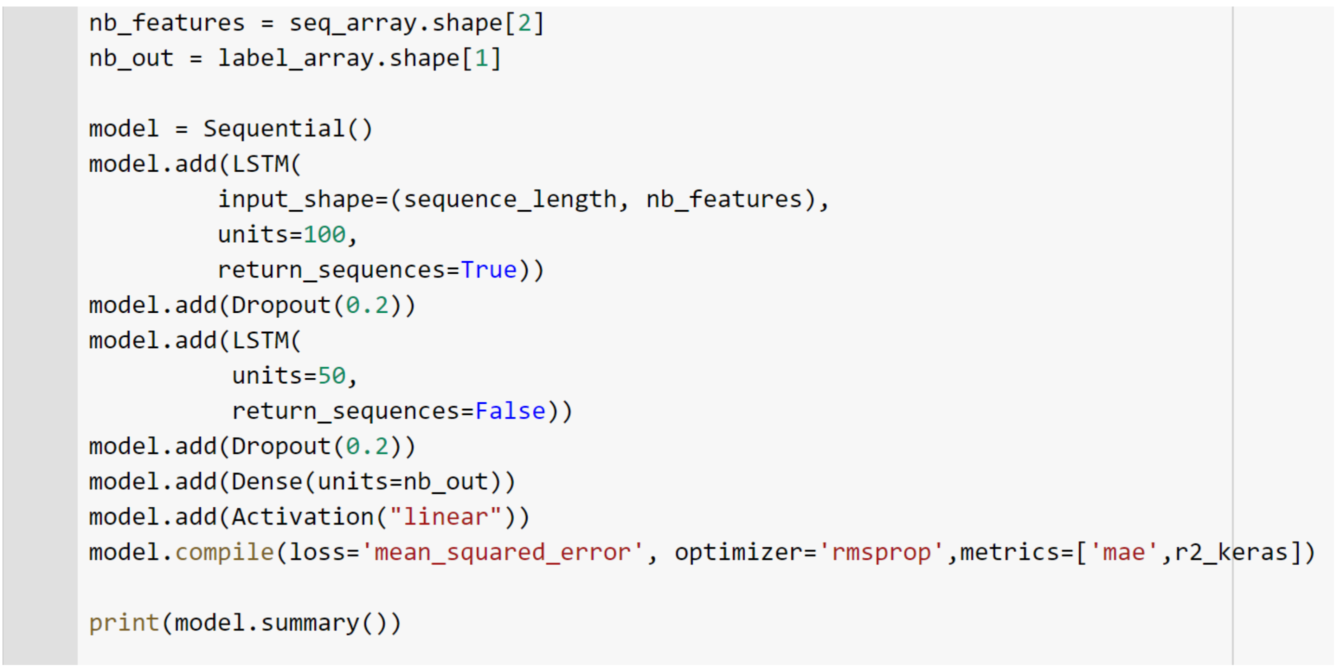
```
[ ]  def r2_keras(y_true, y_pred):
         """Coefficient of Determination
         """
         SS_res =  K.sum(K.square( y_true - y_pred ))
         SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
         return ( 1 - SS_res/(SS_tot + K.epsilon()) )
```

The **coefficient of determination** (denoted by R2) is a key output of regression analysis. It is interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variable.

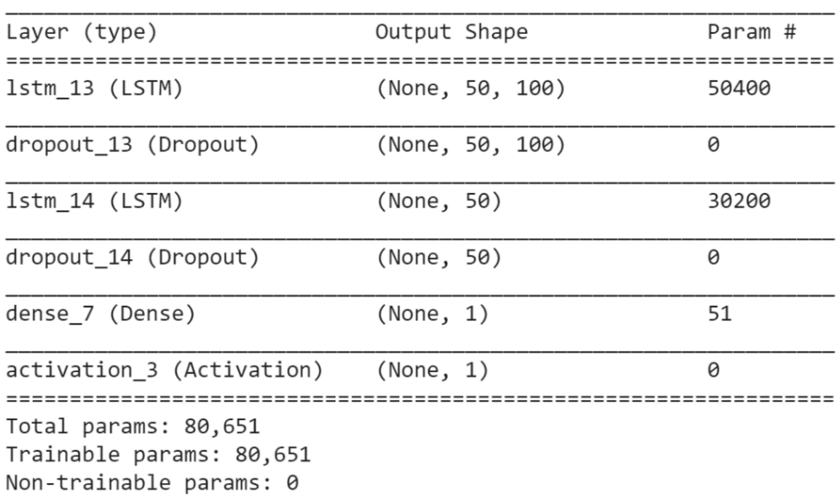
- The coefficient of determination is the square of the correlation (r) between predicted y scores and actual y scores; thus, it ranges from 0 to 1.
- With linear regression, the coefficient of determination is also equal to the square of the correlation between x and y scores.
- An R2 of 0 means that the dependent variable cannot be predicted from the independent variable.
- An R2 of 1 means the dependent variable can be predicted without error from the independent variable.

3. Next, we build a deep network. The first layer is an LSTM layer with 100 units followed by another LSTM layer with 50 units. Dropout is also applied after each LSTM layer to control overfitting.

The final layer is a Dense output layer with a single unit and linear activation since this is a regression problem.



```
nb_features = seq_array.shape[2]
nb_out = label_array.shape[1]

model = Sequential()
model.add(LSTM(
         input_shape=(sequence_length, nb_features),
         units=100,
         return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(
          units=50,
          return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=nb_out))
model.add(Activation("linear"))
model.compile(loss='mean_squared_error', optimizer='rmsprop',metrics=['mae',r2_keras])

print(model.summary())
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_13 (LSTM)               (None, 50, 100)           50400     
_________________________________________________________________
dropout_13 (Dropout)         (None, 50, 100)           0         
_________________________________________________________________
lstm_14 (LSTM)               (None, 50)                30200     
_________________________________________________________________
dropout_14 (Dropout)         (None, 50)                0         
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 51        
_________________________________________________________________
activation_3 (Activation)    (None, 1)                 0         
=================================================================
Total params: 80,651
Trainable params: 80,651
Non-trainable params: 0
```

Loss function Mean squared error (MSE) is the most commonly used loss function for regression. The loss is the mean overseen data of the squared differences between true and predicted values.

RmsProp is an optimizer that utilizes the magnitude of recent gradients to normalize the gradients. We always keep a moving average over the root mean squared (hence Rms) gradients, by which we divide the current gradient.

Stacking LSTM hidden layers makes the model deeper, more accurately earning the description as a deep learning technique. It is the depth of neural networks that are generally attributed to the success of the approach on a wide range of challenging prediction problems.

Additional hidden layers can be added to a Multilayer Perceptron neural network to make it deeper. The additional hidden layers are understood to recombine the learned representation from prior layers and create new representations at high levels of abstraction. For example, from lines to shapes to objects.

A sufficiently large single hidden layer Multilayer Perceptron can be used to approximate most functions. Increasing the depth of the network provides an alternate solution that requires fewer neurons and trains faster. Ultimately, adding depth it is a type of representational optimization.

4. Train the model on the training dataset, show as follows:

```
history = model.fit(seq_array, label_array, epochs=100, batch_size=200,
                    validation_split=0.05, verbose=2,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss',
                       min_delta=0, patience=10, verbose=0, mode='min'),
                       keras.callbacks.ModelCheckpoint(model_path,monitor='val_loss',
                       save_best_only=True, mode='min', verbose=0)]
          )

# list all data in history
print(history.history.keys())
```

5. The trained model provides an r2 value of 0.79 on the test dataset and 0.79 on the validation dataset. We can improve our results by hyper tuning the model parameters. Following, you can see the loss of the model for train and validation datasets during training:

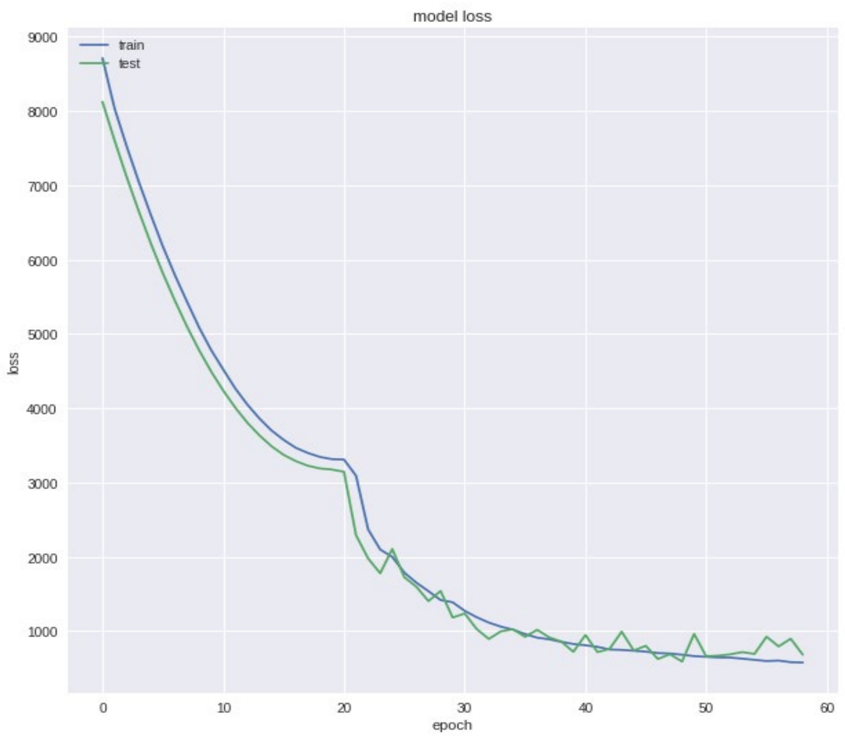


Figure: 4.2 Loss of the trained model

CHAPTER 5 - RESULT ANALYSIS, CONCLUSION & FUTURE WORK

5.1 RESULT

The model accuracy for the Training Set and Validation Set is 98% which is pretty good for a prediction model.

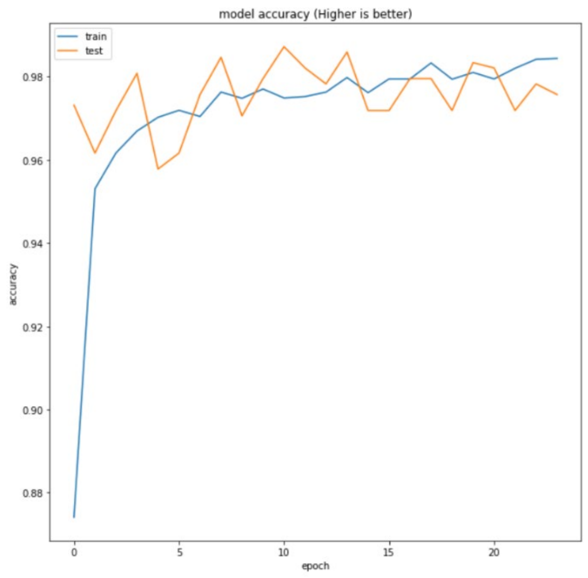


Figure: 5.1 Model Accuracy

The model loss for the Training Set and Validation Set is also very low which is pretty good for a prediction model. And achieving a precision of 96% and an accuracy of 96%.

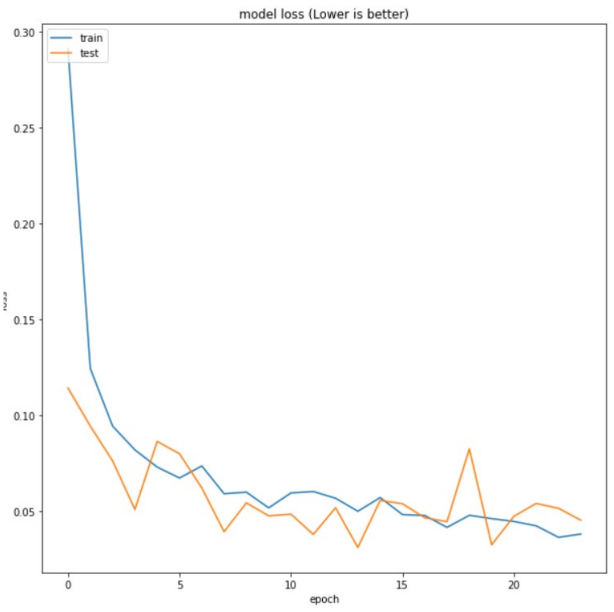


Figure: 5.2 Model Loss

The model R-squared for the Training Set and Validation Set is 79% which is pretty good for a prediction model.

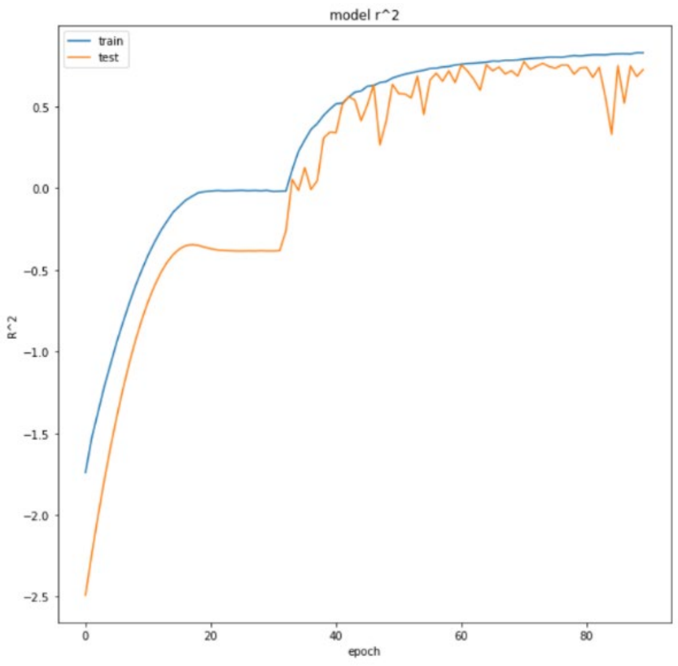


Figure: 5.3 Model r^2

The model for Mean Absolute Error for the Training Set and Validation Set is 14.89 (Lower is better) which is pretty good for a training model.

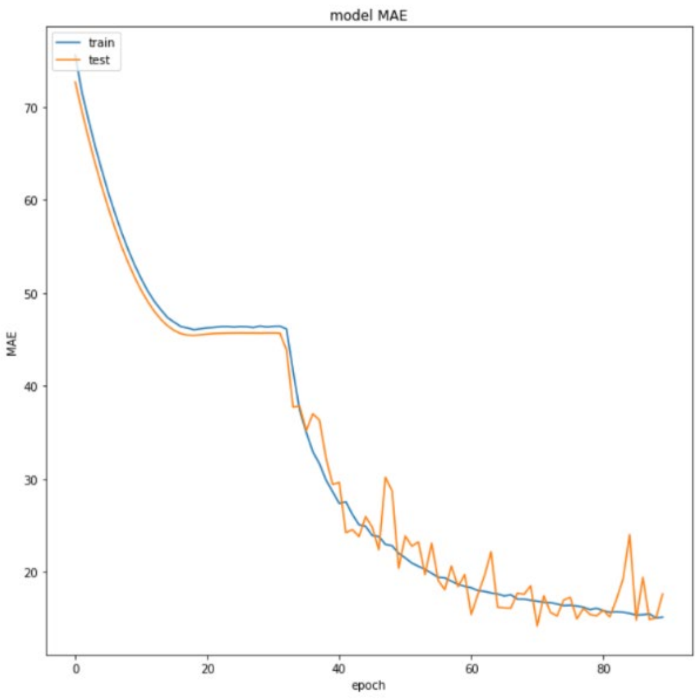


Figure: 5.4 Model MAE

5.2 CONCLUSION

By implying LSTM technique, we were able to get 96% accuracy making one application to be best reliable in predicting RUL of Aircraft Engine and hence by making passenger life safer and predictive maintenance much easier.

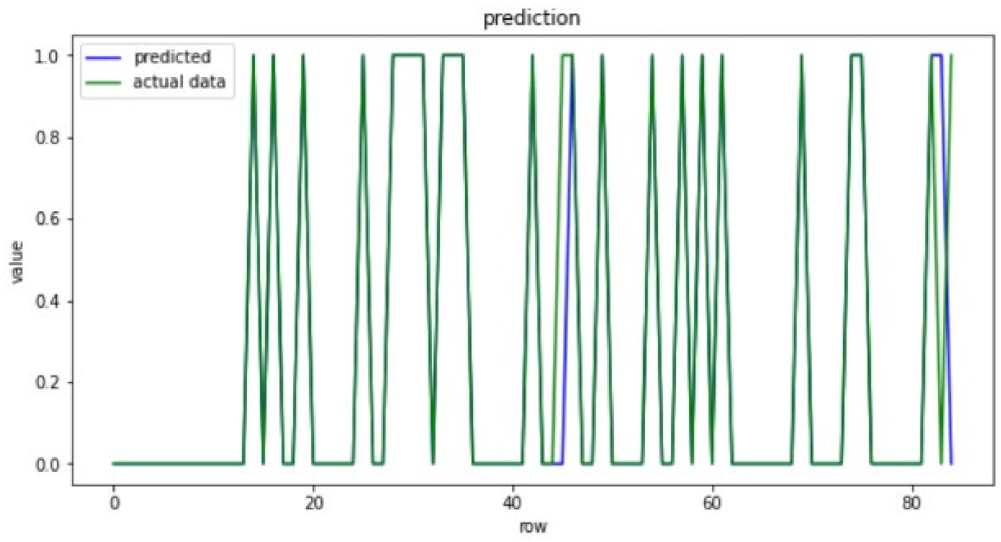


Figure: 5.5 Predicted data vs Actual data

5.3 FUTURE WORK

When machine data can be used to perform predictive maintenance with a high level of precision, manufacturers can focus on differentiating products using digital capabilities like self-awareness of technical health.

A manufacturer's value can be measured not only by the quality of its shop floor processes but also by how it protects its assets. This can be achieved by using predictive maintenance to extend equipment life and improve the efficiency of maintenance procedures.

Predictive maintenance is an essential part of the factory of the future. Manufacturers who automate not only manufacturing processes, but also equipment maintenance, can benefit from a whole new level of production efficiency.

Predictive maintenance is offered as a solution by solution vendors in the market. The solution helps reduce unforeseen downtime with optimized scheduling and failure prevention. It also boosts overall equipment effectiveness by optimizing the production conditions of each machine and process. It minimizes the overall throughput by effectively identifying process bottlenecks. Therefore, the solution acts as a very effective tool for almost all industry verticals.

The predictive maintenance market is driven by a rise in the need among industries such as energy, utility, manufacturing, and automotive among others to minimize the overall operational cost. Moreover, a rise in the use of sensors among these industries is expected to drive the predictive maintenance market. Sensors are being used at the plant site, during distribution, etc.

This aggressive need to be competitive and on schedule in the predictive maintenance market is expected to increase demand for predictive maintenance solutions and services in the coming years. Also, the introduction of a new business model for delivering predictive maintenance techniques to the predictive maintenance market is estimated to propel the predictive maintenance market in the coming years. However, low level of awareness among enterprises about the uses and benefits of predictive maintenance solutions is expected to restrain the predictive maintenance market soon. Nevertheless, with time, the effect of the aforementioned restraint is expected to diminish as more and more companies realize the importance of the solutions.

CHAPTER 6-REFERENCES

[1] Mobley, R Keith, “An Introduction to predictive maintenance”, 2002, 2nd ed, ISBN 0-7506-7531-4

[2] Groba. C, Cech. S, Rosenthal. F., Gossling. An "Architecture of the predictive maintenance framework", 6th International Conference on Computer Information Systems and Industrial Management Applications, 2007, IEEE

[3] M. Yuan, Y. Wu and L. Lin, "Fault diagnosis and remaining useful life estimation of aero engine using LSTM neural network," 2016 IEEE International Conference on Aircraft Utility Systems (AUS), Beijing, 2016, pp. 135-140, Doi: 10.1109/AUS.2016.7748035.

[4] D. Bruneo and F. De Vita, "On the Use of LSTM Networks for Predictive Maintenance in Smart Industries," 2019 IEEE International Conference on Smart Computing (SMARTCOMP), Washington, DC, USA, 2019, pp. 241-248, Doi: 10.1109/SMARTCOMP.2019.00059.

[5] Likhitha and Dr. R. Nagaraja. “Prediction of Remaining Useful Life of an Aircraft Engine Using LSTM Network.” (2019).

[6] Ahmed, M., Baqqar, M., Gu, F., Ball, A.D., 2012. “Fault detection and diagnosis using principal component analysis of vibration data from a reciprocating compressor”, in: Proceedings of the UKACC International Conference on Control, 3-5 September 2012, IEEE Press.

[7] E. Nardo, “Distributed implementation of a LSTM on Spark and Tensorflow”, 2016

[8] Q Zhou, J Son, S Zhou, X Mao, M Salman, Remaining "Useful life prediction of individual units subject to hard failure", IIE Transactions Vol 46, Jun2014, pp. 1017-1030, DOI:10.1080/0740817X.2013.876126

[9] A. Jardine, D. Lin, D. Banjevic. “A review on machinery diagnostics and prognostics implementing condition-based maintenance”, Mechanical Systems and Signal Processing, vol 20, Oct 2006, pp. 1483-1510, DOI: 10.1016/j.ymssp.2005.09.012

[10] A. Jain, P. Kundu, B. K. Lad, “Prediction of remaining useful life of an aircraft engine under unknown initial wear minutes”, 5th International & 26th All India Manufacturing Technology,

Design and Research Conference (AIMTDR 2014), Dec 2014, pp. 494:1- 494:5, DOI: 10.1007/978-81-322-2352-8

[11] A. K. Mahamad, S. Saon, T.. Hiyama. "Predicting remaining useful life of rotating machinery based artificial neural network." Computers & Mathematics with Applications, vol.60, pp. 1078-1087, 2015

[12] Keras: Deep Learning library for Theano and TensorFlow, https://keras.io/

[13] Elephas: Distributed Deep Learning with Keras & Spark, https://github.com/maxpumperla/elephas

[14] D. Dong, X. Li and F. Sun, "Life prediction of jet engines based on LSTM-recurrent neural networks," 2017 Prognostics and System Health Management Conference (PHM-Harbin), Harbin, 2017, pp. 1-6, DOI: 10.1109/PHM.2017.8079264.

[15] Lipton, Zachary. (2015). A Critical Review of Recurrent Neural Networks for Sequence Learning.