Bachelor Thesis Project Report

Prediction of Remaining useful life of an aircraft engine using data driven methodology

SUBMITTED BY

S Uday Bhaskar

16ME10058

UNDER GUIDANCE OF

Dr. Debasis Samanta

Computer Science Department

Contents

Page Number

Data Description

The challenge was focused on data-driven techniques for PHM; no system or domain-specific information was given. The data provided consisted of a multivariate time series for each unit, each of which starts with the unit in normal condition. The variables in each series corresponded to 3 operational settings (defining modes or operational tasks the unit was undertaking) and 21 sensor measurements. At an unspecified point in each series, a fault occurs causing the unit to degrade, resulting in the eventual failure at the end of the series. The data provided was split into training and test sets, where each series in the test set was terminated sometime before the system failure.The task was to estimate the remaining usable life left of an unspecified complex system using a purely data-driven approach.

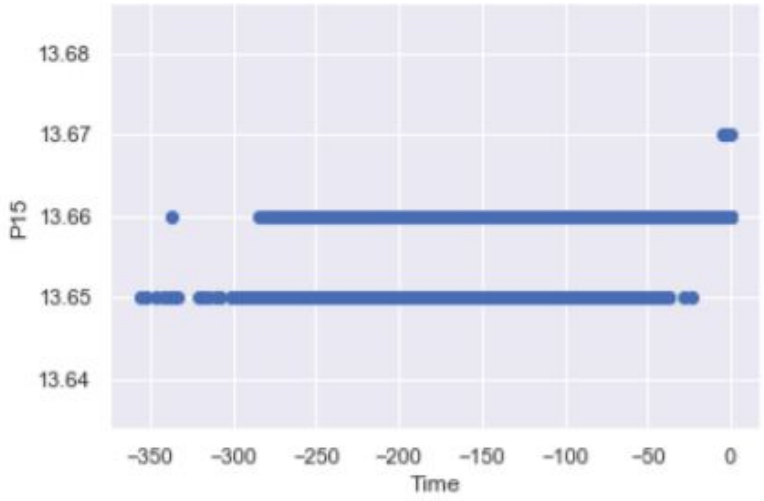
Methodology

I implemented a model with a combination of **Random Forest** for predicting **HI(**Health Index) and an **LSTM**(Long Short Term Memory) neural network for predicting the **RUL**(Remaining Useful Life). **HI** indicates the probability of a given state to be healthy which is achieved by taking probability of predicted outcomes of all decision trees of a random forest.This obtained HI is used as one of the features along with 9 sensors,Time cycle and Operational setting for training LSTM network. This LSTM network takes 50 consecutive time steps, 12 features as input and predicts the last time step's RUL. For prediction,the last 50 cycles of each engine of the test set are considered as input and for engines having less than 50 time cycles an RUL of 125 is assumed. The best score obtained by the above approach is **1131**.

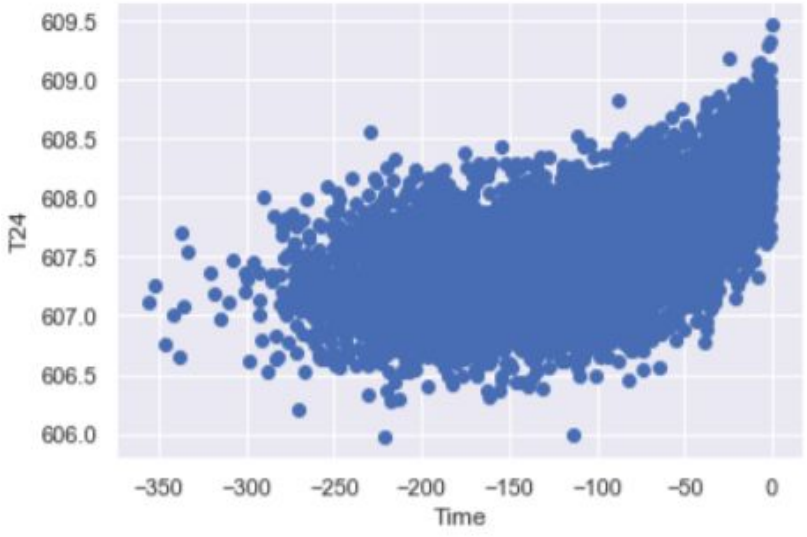
Sensor Selection

Most of the sensors with continuous values exhibit a monotonic trend during the lifetime of the units. However, some of them show inconsistent end-life trends among the different units in the training data set which might indicate, for example, different failure modes of the system. It might be possible to first classify the units by failure modes based on these sensors and then process them using different prediction models; this strategy, however, will encounter two challenges. First, the end-life readings of these sensors spread out over a large range, which makes it hard to quantize the failure modes without extra information. Second, the failure modes might not be unambiguously identifiable, if not completely indiscernible, at the early age of a unit, and thus might contribute little to RUL estimation when only early history of the unit is available. Therefore, only those continuous-value sensors with a consistent trend are selected for further processing. These sensors are indexed by **2, 3, 4, 7, 11, 12, 15.**

From Below graphs, Fig(a) graph of the P15 vs Time cycle shows an example of inconsistent trend of sensors while the Fig(b) graph of T24 vs Time cycle shows an example of consistent trend of sensors.



Fig(a)



Fig(b)

Random Forest

Random forest is a **supervised Learning algorithm** which uses ensemble learning methods for **classification and regression**. “An ensemble method is a technique that **combines the predictions from multiple machine learning algorithms** together to make more accurate predictions than any individual model. A random forest is a meta-estimator (i.e. it combines the result of multiple predictions) which aggregates many decision trees, with some helpful modifications.” The key concepts that are being implemented by a random forest model are

(a) Randomly selecting training data points for building trees

(b) Randomly select a set of features for making a split

For learning the aspects of a problem, a random sample of observations are fed to the random forest model. These samples are drawn with replacement, known as *bootstrapping. Some of these samples will be used during the building of other trees also.* This reduces the chances of having high variance by distributing the data and thereby the variance in the observation among different trees. Additionally, the number of attributes that can be used for building a tree is restricted to some percentage of the total ensuring limited dependence on one particular attribute for making a model. Both these changes include wide diversity in the final model making it better than the individual model. The individual decision tree classifiers are then aggregated to form a random forest ensemble **combining their decision making process and result.** Aggregation of models is done either through model votes or averaging.

Procedure

(a) Randomly pick K number of data points from the training set.

(b) Build the decision tree associated with those K data points.

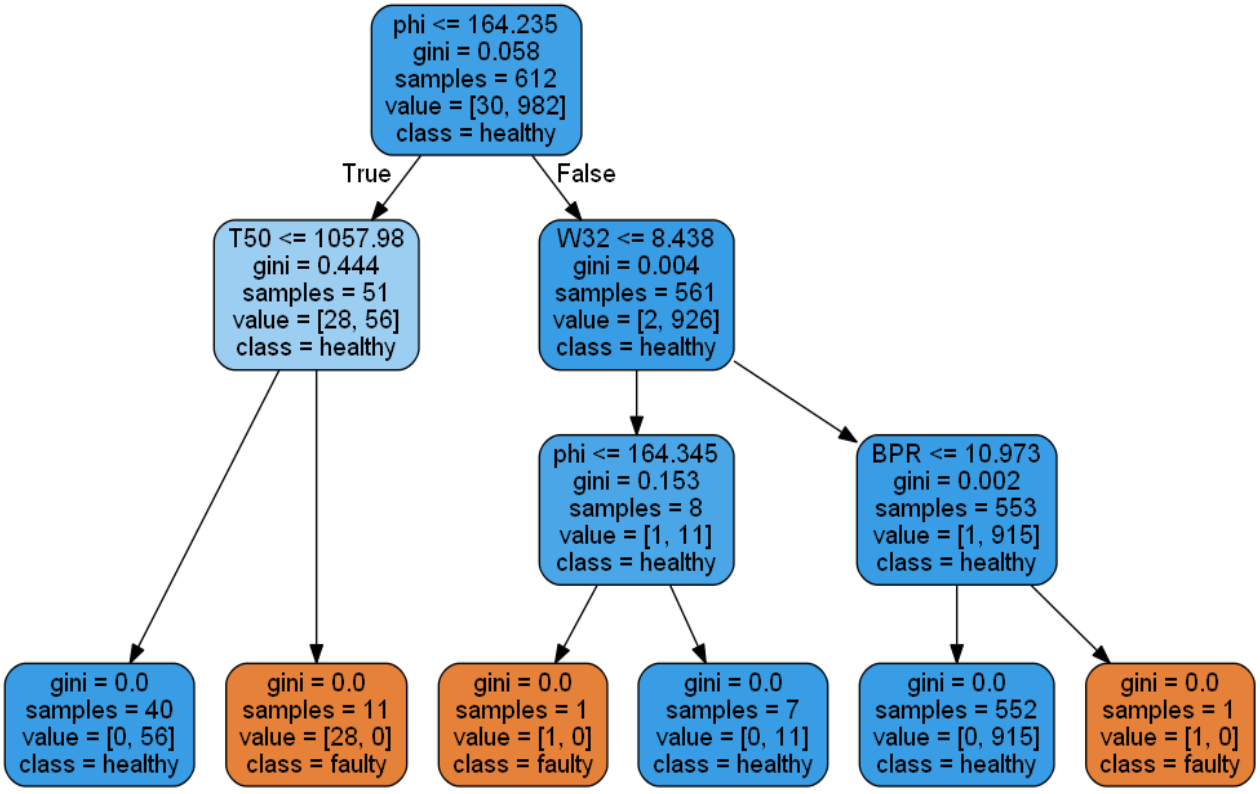
(c) Choose the number of trees (N) you want to build and repeat step 1 & 2.

(d) For any new data point, make predictions using N trees created in step 3 and assign the class by taking the majority vote or average across all the predicted outputs.

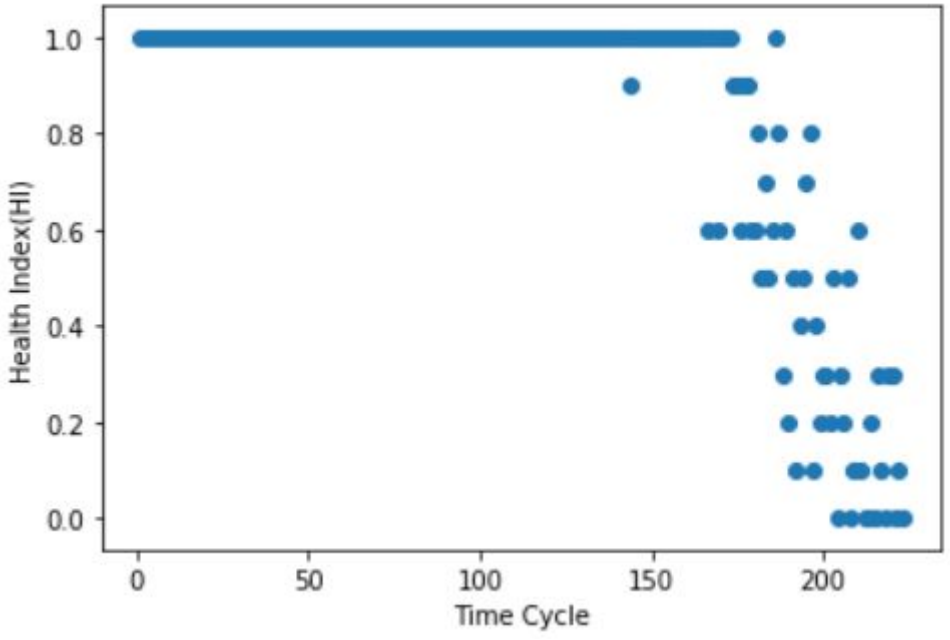
Implementing Random Forest for Health Index Calculation

In this study, the dataset with us has no known value of health index corresponding to sensor values. The only thing known is that the defect grows in size until system breakdowns in the training set. This indirectly means the last time cycle in the training set represents a situation where the engine has reached a failure condition. However, in the test set, the time series ends some time prior to system failure and we have to predict remaining useful life from this point. So, in order to train our model we have to make assumptions regarding the health state of the system. In our approach, we have assumed that the first 30 time cycle of each engine represents a healthy state while the last time cycle represents a failure state. The response or the output value for the healthy state is assumed to be 1 while for the failure state it is 0. The health index between these time cycles essentially has to be between 0 and 1. Based on these assumptions, random forest of trees is formed for the system as per the operating region. Such trees predict the health state of an engine having range between 0 and 1 at each time instance. In a decision tree, split points are chosen by finding the attribute that results into the lowest cost. The cost function is the Gini index. A Gini index of 0 means class values are perfectly separated into two groups, in the case of a two-class classification problem. Finding the best split point in a decision tree involves evaluating the cost of each split in the training dataset for each input variable. Using these sets of 9 sensor values and HI, a number of trees are built. After the model is prepared, it is used to make predictions for other time instance values. Probability of a given sample to be healthy is calculated as no of decision trees predicting it as healthy to the no. of total trees in the forest.
I have implemented the above algorithm in python using various libraries.

Here is an example of a decision tree of a random forest.



Below is the graph of HI vs Time Cycle of an aircraft



Data Pre-Processing for LSTM

Operating Region Partitioning: Partitioning the raw data from the sensors as per the different set of environmental and load conditions (Operating Regime) in which the system had worked during the operational life.

Min-Max Normalization: In this technique of data normalization, linear transformation is performed on the original data. Minimum and maximum value from data is fetched and each value is replaced according to the following formula

x' = (x-min(x))/(max(x)-min(x))
Where x is the attribute data
min(x), max(x) are the minimum and maximum absolute value of x respectively.
x' is the new value of each entry in data.
x is the old value of each entry in data.

This Min-Max Normalization technique was implemented on each Operating region partitioned data of both training and test set.

Long Short Term Memory (LSTM) Neural Network

Since the standard RNN are not suitable for correct prediction when the context information is large, researchers have developed another structure called Long Short term Memory Neural Network (LSTM). LSTM has overcome long-term time dependency problems by controlling flow of input information with the help of input gate, forget gate and output gate. The default behaviour of LSTM is to remember information for long periods of time.

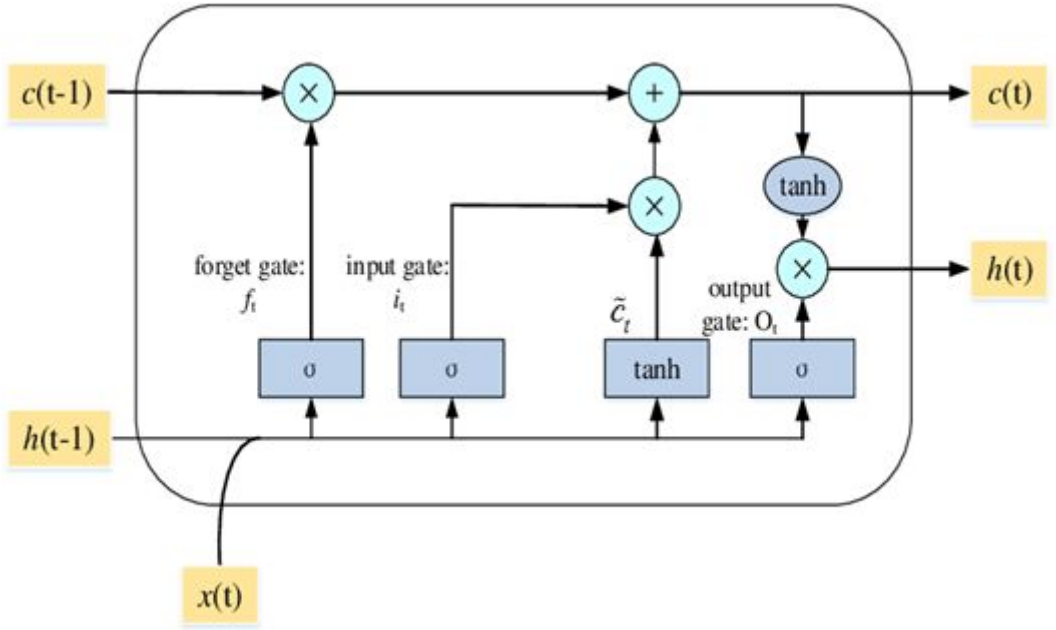
A typical building block of LSTM is shown in figure below. A typical LSTM cell varies from the standard recurrent cell in two ways. Firstly, it divides the state of cell into two parts: short-term state $h(t)$ and the long-term state $c(t)$. Secondly, three gates have been introduced in the cell to control the state path. These gates are forget gate, the input gate and the output gate.

(a) Forget Gate:

The forget gate f(t) decides what information will be thrown out of the last long-term state $c_{(t-1)}$. It reads previous short-term state $h_{(t-1)}$and current state x_t, and outputs a value between 0 and 1 to cell state $c_{(t-1)}$through an activation function. In this case, 1 means "completely keep" and 0 means "completely leave." The f_t can be described as

$$f_t = \sigma\left(w_{xf}^{\top} X_t + W_{hf}\, h_{(t-1)} + b_f\right) = \sigma(f)$$

where (w_{xf}, b_f) are the weight vectors and bias term of the forget gate, σ is the sigmoid activation function.



(b) Long-term state updating The next step is to determine what new information is stored in the long-term cell state. There are two parts. First, the sigmoid layer called the "input gate layer" determines what values we will update. Then, a tanh layer creates a new candidate vector which will be added into the state. The input gate i_t and candidate vector g_t are mathematically expressed as

$$i_t = \sigma\left(W_{xi}^T x_t + W_{hi} h_{(t-1)} + b_i\right)$$
$$\widetilde{c_t} = \tanh\left(Wxc^T x_t + W_{hc} h_{(t-1)} + b_c\right)$$

Where (W_{xi}, Wxc) are the weight vectors, and (*bi*, *bc*) are bias term.

(c) Short-term state updating. The long-term state is filtered through the current output gate to get the short-term state. The output gate is described as

$$\mathbf{O_t} = \sigma\left(W_{xo}^T x_t + W_{ho} h_{(t-1)} + b_o\right)$$

Where W_{xo} is a weight vector and b_o is a bias term.

Then we can obtain a new cell state c_t as

$$c_t = f_t \otimes c_{(t-1)} + i_t \otimes \widetilde{c_t}$$

Then, the short-term state of the LSTM unit at time *t* can be described as

$$h_t = o_t \otimes \tanh(c_t)$$

Where (⊗, ⊕) are element-wise multiplication and addition respectively.

Backpropagation through time or BPTT procedure is used to enable learning of the LSTM network. First delta is computed as $\delta z(t)$ is computed as:

$$\delta z_t = \Delta_t + W_{hg}\,\delta + W_{hi}\,\delta i_{(t+1)} + W_{hf}\,\delta g_{(f+1)}\,W_{ho}\delta O_{(t+1)}$$

where Δ_t is the vector of deltas coming down from the layer above.

The gradients of the weights can be computed by first obtaining the corresponding deltas of the gate. For example, the output gate is computed as:

$$\delta O_t = \delta z_t \otimes \phi(c_t) \otimes' \sigma(o_t)$$

Then, the corresponding gradients for the weights are computed as the sum of outer products, for example, at output gate:

$$\delta Wxo = \sum_{t=0} \langle \delta \overline{Ot}, \mathrm{Xt} \rangle$$

Bi-directional LSTM

The main idea behind any deep neural network architecture is to progressively learn higher levels of abstraction from the input data by performing operation in different layers of the network and thus increase the chances of accurately characterizing the hidden patterns. In a similar way, increasing the number of LSTM layers enables handling strong nonlinearity in the system. To make this possible, a number of LSTM layers are stacked together enabling movement of input data through multiple non-linear LSTM layers. The hidden states are relayed both to the adjacent layer as well as to the LSTM layer immediately above. With the same concept, a bi-directional LSTM structure is constructed wherein flow of data takes place both in forward and backward direction. While the forward flow enables discovering the variations in system parameters, the backward flow allows smoothening of predictions. Both forward and backward paths are calculated independently, however, they are combined together before transmitting to the second bidirectional layer. The equations for the backward path are

$$\underline{f_t} = \sigma(\underline{W_{xf}^T}\,\underline{X_t} + \underline{W_{hf}^T}\,\underline{h_{(t+1)}} + \underline{b_f}) = \sigma(\overline{f_t})$$

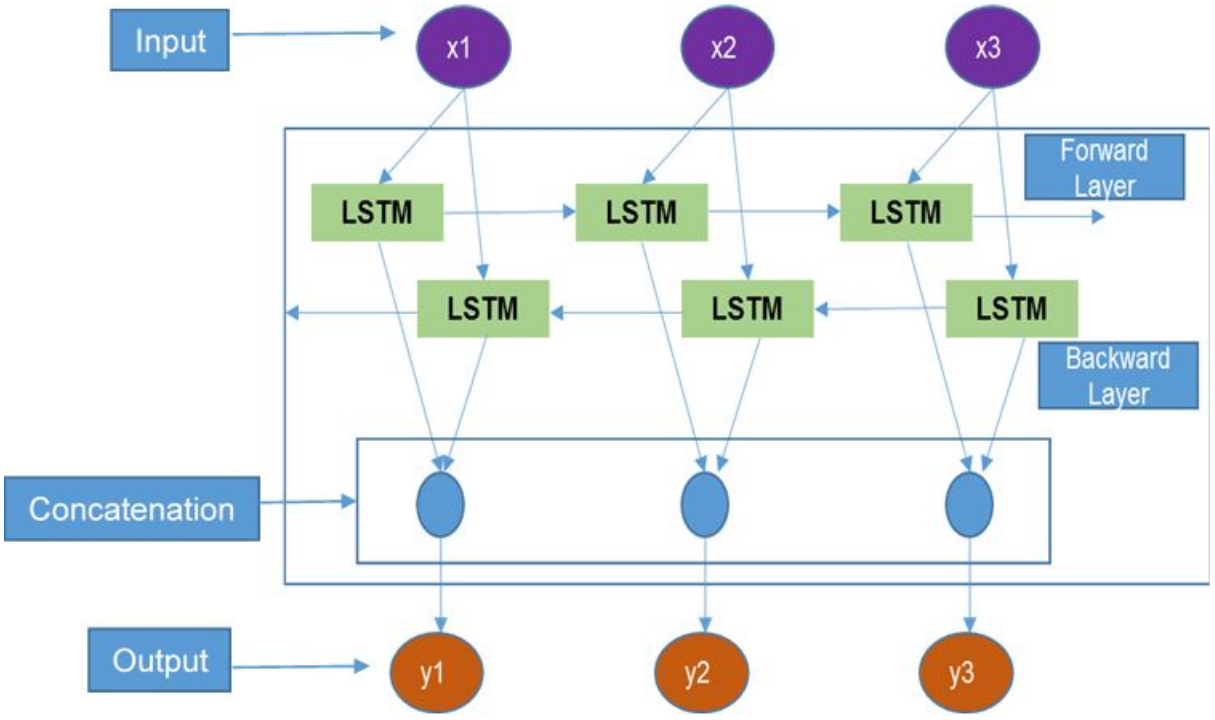
$$\underline{g_t} = \phi(\underline{W_{xg}^T}\,\underline{X_t} + \underline{W_{hg}^T}\,\underline{h_{(t+1)}} + \underline{b_g}) = \phi(\overline{g_t})$$

$$\underline{c_t} = \underline{f_t} \otimes \underline{c_{(t+1)}} + \underline{i_t} \otimes \underline{g_t}$$

$$\underline{h_t} = \underline{o_t} \otimes \phi(\underline{c_t})$$

The underbars in the above equation depict the backward path.

A two layer bi-directional LSTM (BD-LSTM), composed of two LSTM layers is shown in the figure below.



Implementation Of LSTM for predicting RUL

Once the health index for all time cycles wrt each engine is defined using random forest, a BD-LSTM network has been constructed to make prediction of Remaining Useful Life of engines in the test dataset.

The implementation involves three phases: learning phase, validation phase and prediction phase.

(a) **Learning Phase**

Essential aspect of any neural network is the learning phase. The correctness of the result is wholly dependent on the effectiveness of this stage. Let us now understand this stage.

For every aircraft, I made x_train samples considering 50 consecutive time steps like 1st to 50th time cycle,2nd to 51st time cycle etc and respective y_train to be 50th cycle's RUL,51st cycle's RUL etc. Therefore an input sample contains 50 time steps and 12 features(9 sensors,HI,Operational

setting,Time cycle) and the corresponding output sample contains the RUL of the last time cycle.

Architecture implemented by us is based on an LSTM model comprising 2 layers of bidirectional LSTMs (BD-LSTM) followed by a dense layer and an output layer of feed forward neural networks (NNs). Each layer of LSTM contains a chain of repeating cells and is defined by cell structure and number of nodes within the cell. The model type chosen is sequential type which forms a linear stack of layers. The output layer is a dense layer consisting of a regular layer of neurons in a neural network where each neuron receives input from all the neurons in the previous layer, thus are densely connected. For the purpose of this study the first layer of bidirectional LSTM contains 24 nodes and the second layer of bidirectional LSTM contains 10 nodes. Each cell within the LSTM layer has the same structure and parameter values. Proposed structure is adequate for RUL estimation due to the fact it utilizes the complementarity modeling capability of BD-LSTM and NN. Dropout layers are added after each LSTM layer. Dropout is a regularization technique, which aims to reduce the complexity of the model with the goal to prevent overfitting. Using “dropout", one randomly deactivated certain units (neurons) in a layer with a certain probability p from a Bernoulli distribution (ranging from 0 to 1). If we set dropout to 0.5 i.e. half of the activations of a layer to zero, the neural network won’t be able to rely on particular activations in a given feed-forward pass during training. As a consequence, the neural network will learn different, redundant representations; the network can’t rely on the particular neurons and the combination (or interaction) of these to be present. Dropout is only applied during training.

As per the structure of the LSTM, the input needs to be in 3 dimensional matrix form before feeding to the network. The three dimensions are batch size, time steps and dimensional data. For the network having 24 cells at the input layer, a series of 50 consecutive time sequences of 9 selected sensors, TRA, time cycle and HI are supplied as inputs to the network. The output of this network is the RUL value for the 50th training cycle corresponding to each batch. The predicted value of RUL is then compared with the true RUL of training set engines using mean square value as the cost function in our model. In the next step, another set of 50 consecutive time sequence data i.e. from the 2nd to 51st time cycle is fed and RUL for the 51st time cycle is made. The weights are optimized using the “rmsprop” optimizer. Early stopping and model checkpoints are used to capture the parameters of the network which gives the lowest mean squared error. The parameters that can be tuned are batch size, no. of nodes and epochs. As the BD-LSTM is highly non-convex, it is difficult to find an optimal network structure corresponding to global minima.

To find a suboptimal network structure, the network is trained again and again by making changes in tunable parameters and score is checked for each combination. This helps in searching the best network parameter and stabilizes the network robustness against the data variations. The summary of the model is as given below

Input shape (None,50,12)

Layer (type)	Output Shape
bidirectional_1 (Bidirection	(None, 50, 48)
dropout_1 (Dropout)	(None, 50, 48)
bidirectional_2 (Bidirection	(None, 20)
dropout_2 (Dropout)	(None, 20)
dense_1 (Dense)	(None, 1)
activation_1 (Activation)	(None, 1)

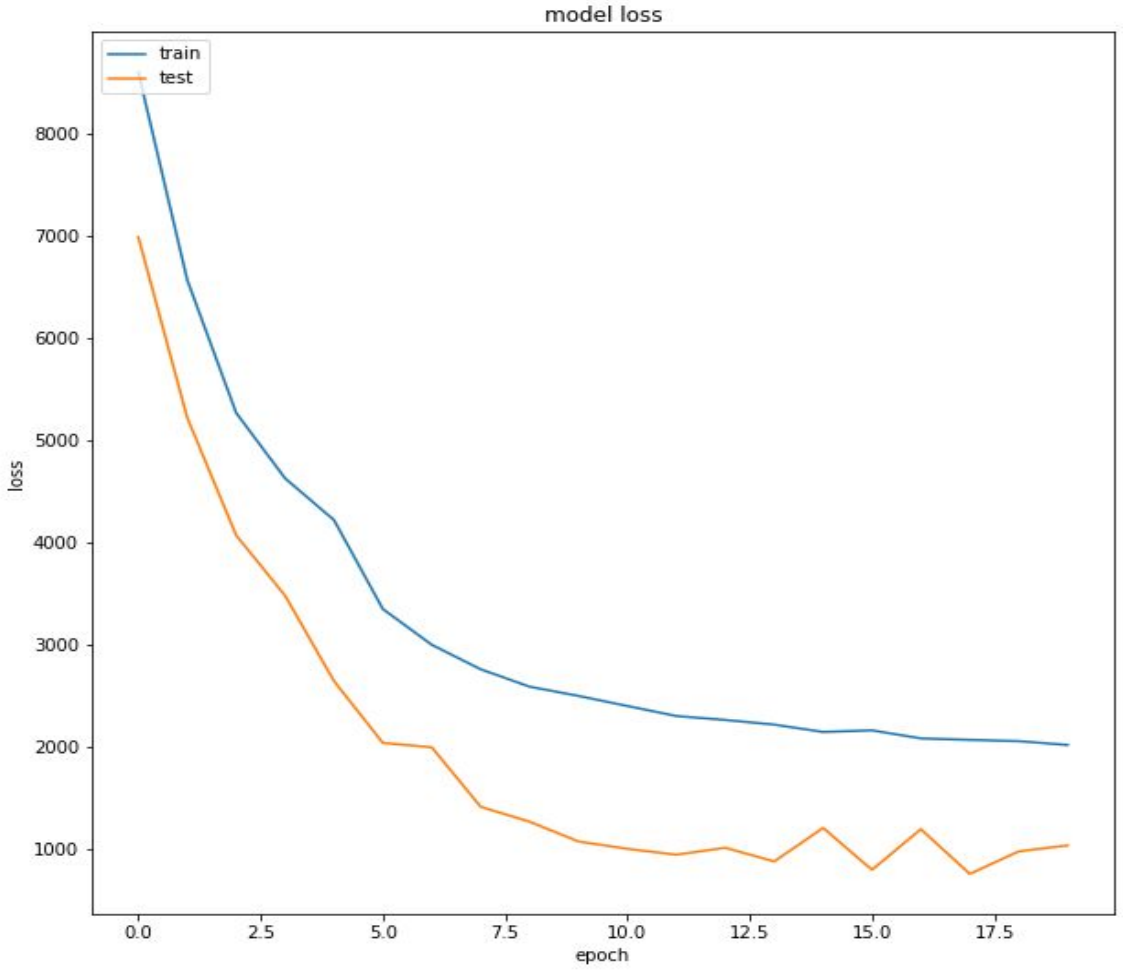
(b) Validation Phase

For each candidate combination of parameters, the complete training set is split randomly into 95% of the sub-training set and 5% of the validation set. Thereafter, using the model learned from the sub-training set, RUL for engines in the validation set is predicted. The difference is calculated between the predicted RUL and the true RUL using mean squared error based cost function. Since the learning in the neural network is a convex problem, the process is repeated 9 times taking a different set validation and training engines during each repetition. The parameter combination that leads to the minimized mean squared error value of RUL predictions in the validation set is selected for final prediction.

(c) Prediction Phase

Preliminary study of the engine data in the test set shows that a lot of engines have less than 50 sequence length (time cycle). The transformation of engines having less than 50 sequence length in the test set data into 3-dimensional form similar to that of training and validation set is not possible. So there is a need for an alternate method. For this, an assumption has been taken that if any engine has a less than 50 time cycle, it's RUL is assumed to be 125. The conviction behind this number is that the preliminary study of the training set has shown that the minimum life of engines is 125 cycles.

Here is a graph of mean squared error(MSE) vs Epochs. Blue line indicates mse of training dataset and orange line indicates mse of validation dataset



Scores and Results

It is not possible to draw a comparison between results obtained using different methods in absolute terms. Therefore, a scoring function has been used for this purpose. It is given as

$$d_i = \text{Estimated } RUL_i - \text{Actual } RUL_i$$

$$s = \begin{cases} \sum_{i=1}^{n} e^{\frac{d_i}{10}} - 1 & for\ d_i < 0 \\ \sum_{i=1}^{n} e^{\frac{d_i}{13}} - 1 & for\ d_i > 0 \end{cases}$$

Where
s is the computed score
n is the number of test units

Clearly, the score of a prediction is defined as an exponential penalty to the prediction error. The score for an algorithm is calculated by summing up the score of each prediction for units in the testing data set. The scoring function is asymmetric around the true RUL values as late predictions will be penalized more than early predictions. In either case, the penalty grows exponentially with increasing error.

The best score obtained by the above approach on the test data is **1131.**

References

[1] Che, Changchang, et al. "Combining Multiple Deep Learning Algorithms for Prognostic and Health Management of Aircraft." *Aerospace Science and Technology* (2019): 105423.

[2] Jianjing Zhang, Peng Wang, Ruqiang Yan, Robert X. Gao, " Long Short Term Memory for machine life prediction", In Journal of Manufacturing Systems 48(2018) pp 78-86, 2018.

[3] Shuai Zheng, Kosta Ristovski, Ahmed Farahat, Chetan Gupta, "Long Short-Term Memory Network for Remaining Useful Life estimation", In IEEE International Conference on Prognostics and Health Management, 2017.