Attention and Long Short-Term Memory Network for Remaining Useful Lifetime Predictions of Turbofan Engine Degradation

Paulo Roberto de Oliveira da Costa, Alp Akcay, Yingqian Zhang, and Uzay Kaymak

School of Industrial Engineering, Eindhoven University of Technology, Eindhoven, Netherlands
p.r.d.oliveira.da.costa@tue.nl, a.e.akcay@tue.nl, yqzhang@tue.nl, u.kaymak@tue.nl

ABSTRACT

Machine Prognostics and Health Management (PHM) is often concerned with the prediction of the Remaining Useful Lifetime (RUL) of assets. Accurate real-time RUL predictions enable equipment health assessment and maintenance planning. In this work, we propose a Long Short-Term Memory (LSTM) network combined with global Attention mechanisms to learn RUL relationships directly from time-series sensor data. We use the NASA Commercial Modular Aero-Propulsion System Simulation (C-MAPPS) datasets to assess the performance of our proposed method. We compare our approach with current state-of-the-art methods on the same datasets and show that our results yield competitive results. Moreover, our method does not require previous degradation knowledge, and attention weights can be used to visualise temporal relationships between inputs and predicted outputs.

1. INTRODUCTION

In machine Prognostics and Health Management (PHM), Remaining Useful Lifetime (RUL) relates to the amount of time left before a piece of equipment cannot perform its intended function. Accurate RUL prognostics enable the interested parties to assess an equipment's health status and to plan future maintenance actions, e.g. logistics of personnel, spare parts and services (Papakostas et al., 2010). In the PHM literature, Physics, Statistical and Machine Learning approaches have been proposed to address the RUL prediction problem. More recently, Machine Learning methods have attracted more attention given their ability to learn without prior information about the degradation mechanisms (Lei et al., 2018).

Machine Learning methods have been studied for RUL prediction, with Neural Networks (NN) receiving much attention given their ability to approximate functions directly from raw data. Recently, Deep NN methods have been proposed to prognostics problems containing high amounts of temporal input data (da Costa et al., 2020). In particular, recent results show that Long Short-Term Memory Networks (LSTM) (Listou Ellefsen et al., 2019) and Convolutional Neural Networks (CNN) (Li et al., 2018) architectures have outperformed traditional prognostics algorithms in RUL predictions for turbofan engine degradation data. Event though LSTMs have shown strong performance on the RUL prediction task, the relationships between inputs and outputs of LSTMs are not easy to interpret. To address this issue, soft attention models (Luong et al., 2015; Bahdanau et al., 2015), offer the promise of providing interpretability of trained weights while retaining the predictive power of Deep Learning approaches.

In our proposed model, we demonstrate that when combined with a variable-level attention mechanism, an LSTM (Hochreiter & Schmidhuber, 1997) can offer transparency directly at the temporal relationship of input variables and output RUL. We show the effectiveness of the proposed method against other methods for the RUL prediction of aircraft engines in the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) datasets. The main contributions of this work are summarised as follows:

- We learn an LSTM model directly from raw multidimensional temporal data.
- We use a soft attention mechanism to provide visualisation of the learned attention weights at each RUL prediction step. The learned weights provide more transparency on the parts of the input important at each prediction step.
- The proposed method achieves high-performance results in several C-MAPPS datasets without incurring in unsupervised pretraining.

The rest of this paper is organised as follows. In the next section, we briefly discuss the state-of-the-art methods for RUL prediction in the C-MAPPS datasets. In section 3, we present our model detailing the learning algorithm and the architecture of our proposed LSTM. In section 4, we present the experimental setup and the selected hyperparameters of our method. Finally, in section 5, we compare and contrast

Paulo R. de Oliveira da Costa et al. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 United States License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

the performance of proposed methods using our datasets and provide analysis of the results.

2. RELATED WORK

In the prognostics literature, several artificial intelligence methods have been proposed to predict the RUL of assets, e.g. linear regression (He & Bechhoefer, 2008), Support Vector Regression (SVR) (Benkedjouh et al., 2013), fuzzy-logic systems (Zio & Di Maio, 2010) and Neural Networks (Tian, 2012). Neural Networks have drawn much attention given their ability to approximate complex functions directly from raw data without information about real degradation (Huang et al., 2007).

However, in many PHM applications, sequential temporal data coming from sensors are the norm. Neural Networks architectures such as Recurrent Neural Networks (RNN) are a natural fit for such problems given that their recurrent internal structure can handle sequential input data. However, due to vanishing gradients, RNNs have issues when learning long-term dependencies (Bengio et al., 1994). To address these issues, Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Cho et al., 2014) networks were introduced. Such networks possess internal gates that control how information flow in the network during learning. Gated architectures enable the network to preserve its memory state over time and reduce the vanishing gradient problem.

In PHM, Yuan et al. (2016) recently showed that LSTMs could outperform RNNs, GRUs and Adaboost-LSTM in an RUL prediction task. Zheng et al. (2017) showed that a sequence of LSTM layers followed by FFNNs could outperform other methods including CNNs in three distinct degradation datasets. Wu et al. (2018) presented similar results by extracting features based on a dynamic difference procedure and later training an LSTM for RUL predictions. Results showed that the LSTM also outperforms simpler RNNs and GRU architectures under similar machinery conditions. More recently, Listou Ellefsen et al. (2019) showed that Restricted Boltzmann Machines could be used to extract useful weight information by pretraining on degradation data in an unsupervised manner. In this two-stage method, weights extracted in the first step are then used in a further step to fine-tune a supervised LSTM and FFNN model. A genetic algorithm (GA) is used to select the best performing hyperparameters. The methodology holds the state-of-the-art prediction results for the C-MAPSS datasets, presenting it as an effective method for temporal degradation data prediction.

CNNs are notable for being able to extract spatial information from 2D and 3D data (Hossain et al., 2019). CNNs can also handle 1D sequential data and extract high-level features by combining convolution and max-pooling operations while sliding a local receptive field over input features. In machine prognostics, Babu et al. (2016) proposed a 2D deep CNN to predict the RUL of a system based on normalised variate time series from sensor signals; they show the effectiveness of the CNN in comparison to Multi-Layer Perceptron (MLP), SVR and Relevance Vector Regression (RVR). Li et al. (2018) proposed to apply 1D convolutions in sequence without pooling operations. The results show that the proposed architecture can extract features useful for RUL prediction. They show competitive results on the C-MAPPS dataset without incurring in high training times encountered when training recurrent models.

Attention is a popular mechanism used in a wide range of neural network architectures. It was originally introduced in Natural Language Processing (NLP) for machine translation tasks (Bahdanau et al., 2015), but has been successfully applied in other tasks, e.g. computer vision (Xu et al., 2015). In such tasks, we often are interested in focusing the attention of the learning network in parts of the inputs instead of the whole input sequence. Besides performance gain, attention mechanisms can also be used as a tool for interpreting the behaviour of neural architectures, by analysing the parts of the input that the network learns to attend (Galassi et al., 2019). Moreover, attention weights learned from data can be used to visualise and investigate the relationship between inputs and outputs of neural network architectures.

Recent results show that LSTMs and CNNs architectures have outperformed traditional prognostics algorithms in RUL predictions for turbofan engine degradation data. However, CNNs do not maintain temporal information, and although LSTMs can maintain temporal information on its hidden states, they require unsupervised pretraining and considerable hyperparameter search to achieve state-of-the-art results on unseen data. Thus, in this work, we propose an LSTM architecture with attention mechanisms capable of learning RUL degradation. We learn our model directly on the input data without requiring feature engineering or unsupervised pretraining method. Later, we visualise the learned attention weights to interpret the behaviour of the network.

3. METHODOLOGY

In this section, we present our proposed LSTM method to predict the RUL of aircraft engines simulated data. We first introduce the problem and the notations and then further discuss the proposed method and its components.

3.1. Problem Definition

We denote our training data pairs as $\{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^N$, containing N training examples. Where $\mathbf{x}^i$ denotes a multivariate time-series input of length T_i and q features, i.e. $\{\mathbf{x}^i = (x_t^i)_{t=1}^{T_i}\} \in \mathbb{R}^{q \times T_i}$. Moreover, $\mathbf{y}^i$ denotes the RUL values of length T_i where $\{\mathbf{y}^i = (y_t^i)\} \in \mathbb{R}_{\geq 0}^{T_i}$. Where for each $t \in \{1, 2, ..., T_i\}$, $x_t^i \in \mathbb{R}^q$ and $y_t^i \in \mathbb{R}_{\geq 0}$ represent the t-th measurement of

sensor inputs and RUL labels. We aim to learn a function g such that we can approximate the corresponding RUL at testing time directly from degradation data, i.e. $\hat{\mathbf{y}}^i \approx g(\mathbf{x}^i)$.

3.2. Time Windows Processing

To allow temporal sequences influence the RUL prediction at a point in time we apply a time window approach for feature extraction. The sequential input is assumed to be $\mathbf{x}^i = (x_t^i)_{t=1}^{T_i}$ where T_i denotes the size of each sequence length. We define a function ϕ that divides each sequence of size T_i in sequential time windows of size T_w, i.e. $\phi_t(\mathbf{x}^i) = (x_{t-T_w+1}^i, ..., x_t^i)$. After the transformation, at time t all previous sensor data within the time window $\phi_t(\mathbf{x}^i)$ are collected to form a high-dimensional input vector used to predict y_{t+1}^i. Thus, after the transformation each original time series will have $n_i = T_i - T_w$ training samples.

3.3. Long Short-Term Neural Networks

LSTMs have recurrent connections capable of learning the temporal dynamics of sensor data in prognostics scenarios. Moreover, they control how information flows within the LSTM cells by updating a series of gates capable of learning long-term relationships in the input data.

In our LSTM implementation, the memory cell (Figure 1) consists of three non-linear gating units that update a cell state $C_t \in \mathbb{R}^l$, using a hidden state vector $h_{t-1} \in \mathbb{R}^l$ and inputs $x_t^i \in \mathbb{R}^q$, where l is the dimension of the LSTM hidden state and q the input dimension:

$$f_t = \sigma\left(W_f x_t^i + R_f h_{t-1} + b_f\right) \tag{1}$$

$$i_t = \sigma\left(W_i x_t^i + R_i h_{t-1} + b_i\right) \tag{2}$$

$$o_t = \sigma\left(W_o x_t^i + R_o h_{t-1} + b_o\right) \tag{3}$$

where σ is a sigmoid activation function responsible for squeezing the output to the 0-1 range, $W_g \in \mathbb{R}^{l \times q}$ are the input weight matrices, $R_g \in \mathbb{R}^{l \times l}$ are the recurrent weight matrices, and $b_g \in \mathbb{R}^l$ are bias vectors. Where the subscript g can either be the forget gate f, input gate i or the output gate o, depending on the activation being calculated.

After computing f_t, i_t and $o_t \in \mathbb{R}^l$, the new cell state $\widetilde{C}_t$ candidate is computed as follows:

$$\widetilde{C}_t = tanh\left(W_C x_t^i + R_C h_{t-1} + b_C\right) \tag{4}$$

where, $tanh$ represents the hyperbolic tangent function and similar to the gate operations: $W_C \in \mathbb{R}^{l \times q}$, $R_C \in \mathbb{R}^{l \times l}$, and $b_C \in \mathbb{R}^l$.

The previous cell state C_{t-1} is then updated to the new cell state C_t:

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \widetilde{C}_t \tag{5}$$

where $\otimes$ denotes the element-wise multiplication.

In other words, in the previous equations, the *forget* gate f_t is responsible for deciding which information will be thrown away from the cell state. Next, the *input* gate i_t decides which states will be updated from a candidate cell state. The input and forget gates are then used to update a new cell state for the next time step.

Lastly, the *output* gate o_t decides which information the cell will output and new hidden state h_t is computed by applying a $tanh$ function to the current cell state times the output gate results.

$$h_t = o_t \otimes tanh(C_t) \tag{6}$$

3.4. Attention Mechanism

Our global attention mechanism is based on the implementation of Luong et al. (2015). At each time step t we take as input the hidden states h_t at the top layer of stacked LSTM layers. After, we decide on a context vector c_t that captures relevant information about the next target y_{t+1}.

Given the target hidden state h_t and the context vector c_t, we use a concatenation layer to combine the information from both vectors and learn $W_c \in \mathbb{R}^{c \times 2l}$, via a Fully-Connected layer of size c to produce an attention vector a_t of the form:

$$a_t = tanh(W_c[c_t; h_t]) \tag{7}$$

Where the context vector c_t is defined as:

$$c_t = \sum_{j=1}^{t} \alpha_{t,j} h_j \tag{8}$$

In other words, we consider all the hidden states of the LSTM encoder weighted by attention weights $\alpha_{t,j}$. In our implementation, the attention weights are derived by comparing the current hidden state h_t with the complete sequence hidden states h_j, $j = 1, .., t$. Where the attention weights $\alpha_{t,j}$ are defined as:

$$\alpha_{t,j} = \frac{exp(e(h_t, h_j))}{\sum_{j=1}^{t} exp(e(h_t, h_j))} \tag{9}$$

and $e(h_t, h_j)$ is given by the multiplicative equation (Luong et al., 2015):

$$e(h_t, h_j) = h_t^{\mathrm{T}} W h_j \tag{10}$$

where $W \in \mathbb{R}^{l \times l}$. Thus, given attention weights $\alpha_{t,j}$ the context vector c_t is computed as the weighted average over all the hidden states and attention vectors a_t are passed to the Fully-Connected (Dense) layers in the network.

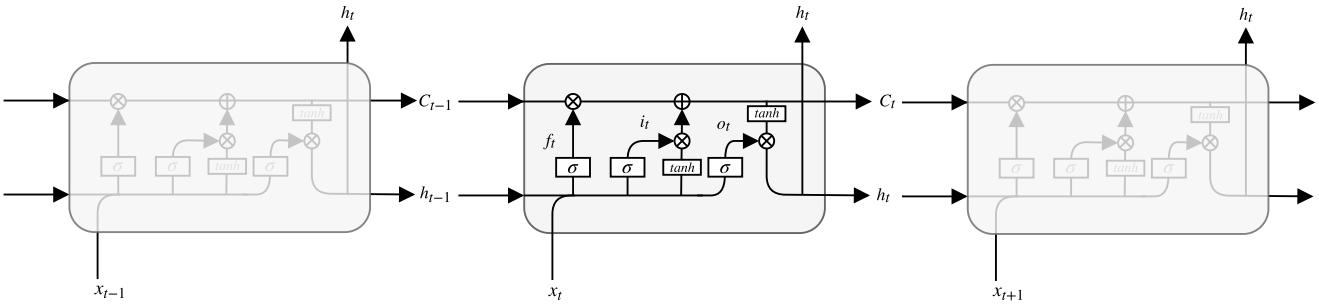


Figure 1. LSTM memory cell.

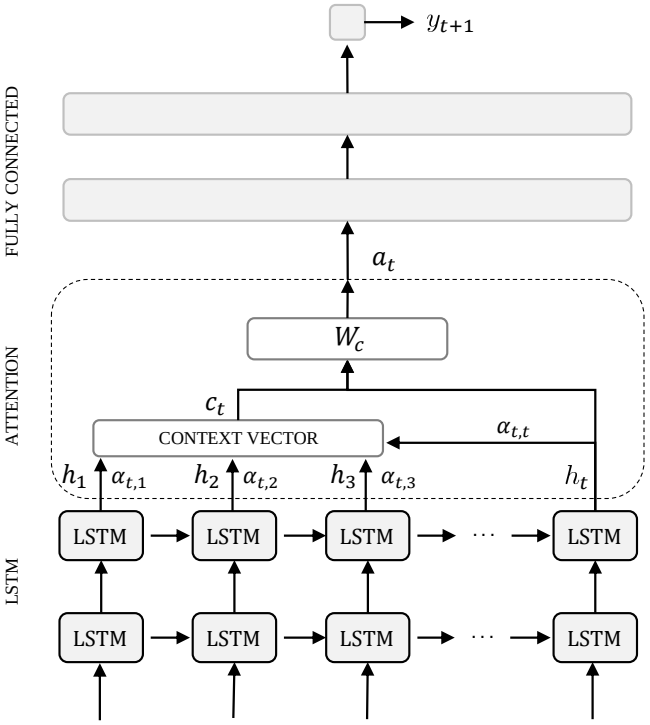


Figure 2. LSTM Architecture with global attention. At each time step t, the model infers a variable-length alignment weight vector a_t based on the current target state h_t and a context vector c_t based on all previously seen states.

3.5. Loss Function

During training, we aim at minimising a regression loss $\mathcal{L}$ using the observed RUL at time t and inputs between $t-T_w+1$ and t. The parameters of LSTM are optimised towards minimising a regression loss function $\mathcal{L}^i$ for each training example of weights θ in the network.

$$\mathcal{L}^i(\theta) = ||\hat{y}^i_{t+1} - y^i_{t+1}||^2 \tag{11}$$

The losses for each *batch* of training examples are then averaged. The loss function errors are passed to weights of the networking using the Back-propagation Through Time (BTT) algorithm. Lastly, weights are optimised using the Adam algorithm (Kingma & Ba, 2015). The proposed architecture is shown in Figure 2.

4. DESIGN OF EXPERIMENTS

In this section, we describe the experiments using the proposed model to predict the RUL of turbofan engine degradation data. We describe the datasets used in the experiments and the details about the implementation.

4.1. C-MAPPS Datasets

The proposed method is evaluated using the benchmark Commercial Modular Aero-Propulsion System Simulation (C-MAPPS) (Saxena et al., 2008) datasets containing turbofan engine degradation data. The C-MAPPS datasets are composed of four distinct datasets that contain information coming from 21 sensors as well as 3 operational settings. Each of the four datasets possesses several degradation engines split into training and testing data. Moreover, the datasets contain run-to-failure information collected under various operating conditions and fault modes.

Engines in the datasets are considered to start with various degrees of initial wear but are considered healthy at the start of each record. As the number of cycles increases the engines begin to deteriorate until they can no longer function. Unlike the training datasets, the testing datasets contain temporal data that terminates some time before a system failure.

The original prediction task is to predict the RUL of the testing units using the training units (Saxena et al., 2008). The details about the four datasets are given in Table 1. We refer to the datasets as FD001, FD002, FD003 and FD004. The operating conditions in the datasets vary between one (sea level) in FD001 and FD003, to six, based on different combinations of altitude (0 - 42000 feet), throttle resolver angle (20 - 100) and Mach (0 - 0.84) in FD002 and FD004. Also, fault modes vary between one (HPC degradation) in FD001 and FD002 and two (HPC degradation and Fan degradation) in FD003 and FD004.

Data	FD001	FD002	FD003	FD004
Engines: Training (N)	100	260	100	249
Engines: Testing	100	259	100	248
Operating Conditions	1	6	1	6
Fault Modes	1	1	2	2

Table 1. The C-MAPPS datasets. Each dataset contains a number of training engines (Engines: Training (N)) with *run-to-failure* information and a number of testing engines (Engines: Testing) with information terminating before a failure is observed.

4.2. Data Preprocessing

The temporal input data coming from 21 sensor values and 3 operational settings are used across the experiments. We note that for both FD001 and FD003 datasets, 7 sensor values have constant readings and have little impact in predicting target RUL values.

We normalise the input data and RUL values by scaling each feature individually such that it is in the (0-1) range using the min-max normalisation method:

$$\bar{x}_t^{i,j} = \frac{x_t^{i,j} - \min(x^j)}{\max(x^j) - \min(x^j)} \tag{12}$$

where $x_t^{i,j}$ denotes the original i-th data point of the j-th input feature at time t and x^j the vector of all inputs of the j-th feature.

In our datasets, RUL targets are only available at the last time step for each engine in the test datasets. However, it is reasonable to estimate the RUL as a constant value when the engines operate in normal conditions (Heimes, 2008). Similar to the works of Listou Ellefsen et al. (2019) and Lei et al. (2018), we propose to use a piece-wise linear degradation model to define the correct RUL values in the training datasets. That is, after an initial period with constant RUL values, we assume that the RUL targets decrease linearly as the number of observed cycles progresses. We denote as R_e the initial period in which the engines are still working in their desired conditions. To allow comparison to previous works, a constant R_e of 125 cycles is selected in our experiments.

4.3. Performance Metrics

Similar to other prognostic studies using the same datasets, we measure the performance of the proposed method of target datasets using two metrics. We propose to use the Root Mean Squared Error (RMSE) as this can be directly related to Eq. (11).

Moreover, we evaluate our model using a scoring function shown in Eq. (13) proposed by Saxena et al. (2008):

$$s = \begin{cases} \sum_{i=1}^{n} e^{-\frac{c_i}{a_1}} - 1, & \text{if } c_i < 0 \\ \sum_{i=1}^{n} e^{\frac{c_i}{a_2}} - 1, & \text{if } c_i \geq 0 \end{cases} \tag{13}$$

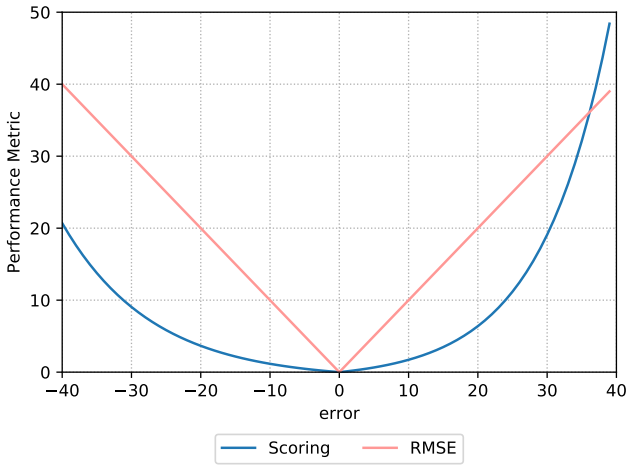


Figure 3. Performance metrics plot. The Scoring performance metric overpenalises positive errors of the RUL prediction.

where $a_1 = 13$ and $a_2 = 10$ and $c_i = \hat{RUL}_i - RUL_i$ (Saxena et al., 2008). That is, c_i is the difference between predicted and observed RUL values. The scoring metric penalises positive errors more than negative errors as these have an impact on RUL prognostics tasks as it can be seen in Figure 3.

4.4. Hyperparameter Analysis

To choose the network architecture for the C-MAPSS data, we performed 10-fold cross validation to estimate the performance of the models. We randomly split units in the original training datasets into training and cross-validation (used for stopping criteria) datasets containing 90% and 10% engines of the original dataset. For example, for FD001, 90 engines are selected for training and 10 for cross-validation.

Hyperparameter	Range
Learning rate	{0.001, 0.01, 0.1}
Batch size	{256, 512, 1024}
Number of layers (LSTM, Dense)	{1, 2}
Number of neurons (LSTM)	{20, 32, 64, 100}
Number of neurons (Dense)	{20, 30}
Number of neurons (Attention)	{128}
L_2 Regularisation	{0.0, 0.01, 0.1}
T_w	{15, 20, 30, 40}

Table 2. Hyperparameter values evaluated in the proposed methodology.

We present the most sensitive hyperparameters: number of LSTM neurons, number of LSTM layers, size of time window T_w and batch size and report report the average RMSE on the test datasets for each dataset in Figures 4-7. For each hyperparameter test we start with a network of the form LSTM(32) + DROPOUT(0.5) + RELU(DENSE(30)) + DROPOUT(0.1) + RELU(DENSE(20)) + DENSE(1), and

proceed to perform grid-search on the individual parameters. In our notation, a learning layer in the network is denoted as ACTIVATION(LAYER(UNITS)), dropout (Srivastava et al., 2014) layers as DROPOUT(RATE) and RELU represents the Rectified Linear Unit Function. In our tests, the remaining parameters of the model, i.e. size of Dense layers, L_2 regularisation and dropout rate were chosen based on previous architectures (Li et al., 2018; Listou Ellefsen et al., 2019) considering the values in Table 2. We train the network for 200 epochs using the Adam (Kingma & Ba, 2015) optimiser with a learning rate of 0.001 and batch size of 256 and select T_w equal to 30, 20, 30, 20 for FD001, FD002, FD003 and FD004 respectively.

Time Window In our experiments, the time window T_w was the most sensitive hyperparameter. In the tests, we considered values of T_w in $\{15, 20, 30, 40\}$. Results in Figure 4 show that for FD001 and FD003 a T_w of 30 timesteps yields the best performing scores, 14.0 and 13.4 respectively. For the remaining two datasets, the lowest RMSE values are found for a T_w of size 20. FD002 has its lowest RMSE at 17.4 and FD004 at 19.4. Moreover, a change in the time window results in RMSE up to 28.4 for FD004 and 20.6 for FD002. It should be noted that in the testing sets the shortest length for datasets FD001, FD002, FD003, FD004 are 31, 21, 38 and 19. Thus, the T_w considered in these experiments takes into account the shortest available length in the test datasets.

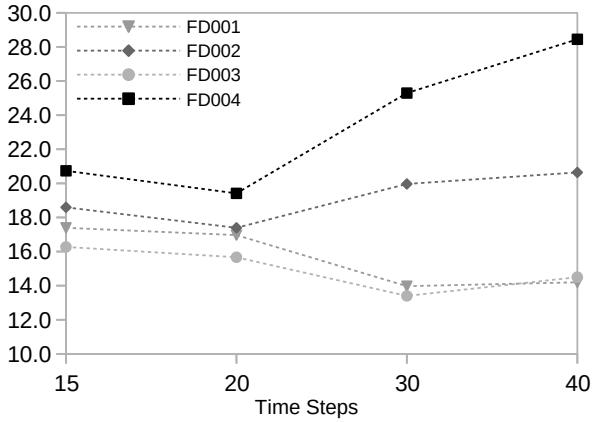


Figure 4. RMSE on the test datasets for different values of T_w.

Batch Size The batch size has also an important effect on the trained model as it affects performance and training speed. Results in Figure 5 show that increasing the batch size does not lead to increased performance. In fact, on average, a batch size of 256 results in the lowest RMSE values across all datasets. Therefore, to allow for more stochastic moves during gradient descent we select a batch size of 256 for our final models.

Number of LTSM Neurons The number of neurons in the LSTM layer has different results depending on the studied dataset shown in Figure 6. For example, for FD004, 32 neurons results in the lowest RMSE at 19.5. However, the variation in performance between 32, 64 and 100 neurons is small.Since 100 neurons result in reasonable performance across all four datasets, we select 100 neurons as the best performing value.

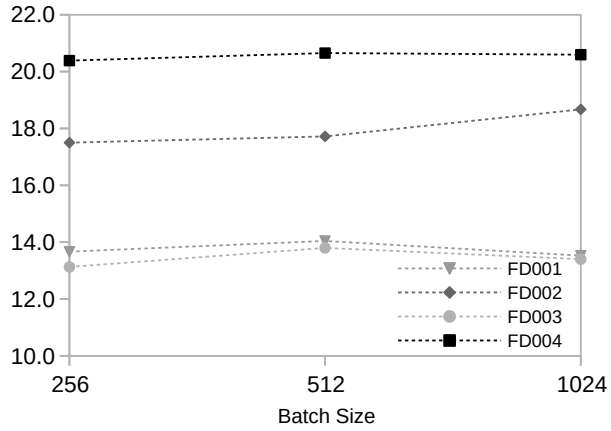


Figure 5. RMSE on the test data for different values of batch size.

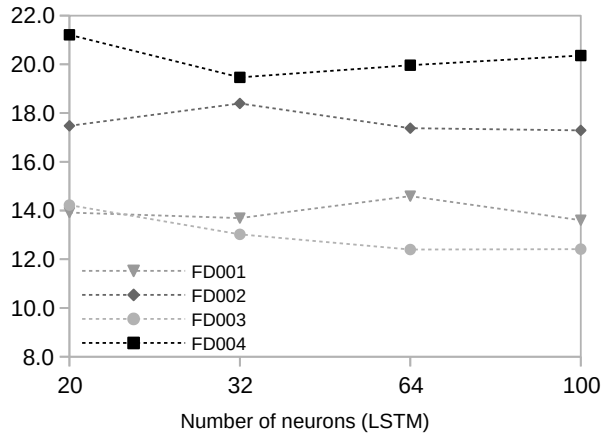


Figure 6. RMSE on the test data for different number of neurons in the LSTM layer.

Number of LTSM Layers We test up to two LSTM layers in the results in Figure 7. In our experiments, adding more LSTM layers did not result in better performance. As we also perform dropout, adding more layers only added more computational burden without further performance gains. Having just one layer containing 100 neurons resulted in the best overall results across all datasets.

	RMSE				
Method	FD001	FD002	FD003	FD004	R_e
LSTM + FFNN (Zheng et al., 2017)	16.14	24.49	16.18	28.17	130
MODBNE (Zhang et al., 2017)	15.04	25.05	12.51	28.66	-
CNN + FFNN (Li et al., 2018)	12.61	22.36	12.64	23.31	125
GA + LSTM (Listou Ellefsen et al., 2019)	**12.56**	22.73	**12.10**	22.66	115-135
Proposed LSTM + FFNN ($\pm$ StDev)	13.64 ($\pm$ 0.80)	17.76 ($\pm$ 0.43)	12.49 ($\pm$ 0.29)	21.30 ($\pm$ 1.06)	125
Proposed LSTM + A ($\pm$ StDev)	13.95 ($\pm$ 0.43)	**17.65** ($\pm$ 0.47)	12.72 ($\pm$ 0.73)	**20.21** ($\pm$ 0.63)	125

Table 3. RMSE comparison between the proposed LSTM methods and other methods in the literature on the C-MAPPS datasets

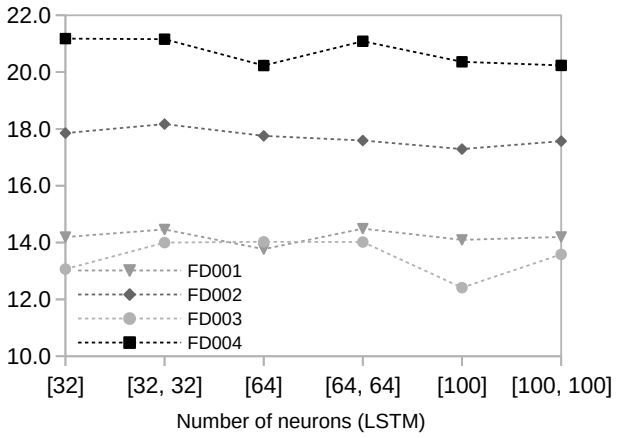


Figure 7. RMSE on the test data for different number of LSTM layers and neurons.

4.5. Training Parameters

Based on the results obtained from the previous section, we select the following architecture, LSTM(100) + DROPOUT(0.5) + ATTENTION(128) + RELU(DENSE(30)) + DROPOUT(0.1) + RELU(DENSE(20)) + DENSE(1). Similar to our hyperparameter search, we split the data into training and cross-validation datasets containing 90% and 10% engines of the original training dataset. To reduce the effect of randomness we perform 10 experiments for each dataset and average the results.

Moreover, the inputs and RUL outputs are normalised individually according to Eq. (12) and the time window transformation ϕ_t is applied. L_2 regularisation is applied in the weights θ in Eq. (11), while dropout layers are used after each LSTM layer to control overfitting. We train the models for a maximum of 200 epochs and select mini-batches of 256 samples for gradient updates. We stop training if no improvement is seen for 20 epochs in the cross-validation dataset. We train the network for 200 epochs using the Adam algorithm, clipping the gradient norms to 1 and using a batch size of 256. We select the learning rate of 0.001 based on grid-search after the remaining architecture has been defined. Finally, we select T_w equal to 30, 20, 30, 20 for training in FD001, FD002, FD003 and FD004 respectively.

We evaluate our models using the C-MAPPS testing datasets, where the goal is to predict the RUL of input sequences seem up to a point before failure. For testing, each RUL label in the testing datasets is provided. In our results, we use rectified labels based on the value $R_e = 125$ for training, validation and testing. That is if RUL values are above R_e they are set to 125. To be able to compare our model outputs to the rectified RUL in the test datasets we multiply the outputs of our LSTM model by R_e to retrieve their original scale.

All our experiments are performed on an Intel Core i5 7th generation processor with 16 GB RAM and a GeForce GTX 1070 Graphics Processing Unit. We implement the models using the Python 3.6 programming language and the Keras (Chollet et al., 2015) deep learning library with TensorFlow (Abadi et al., 2015) backend.

5. RESULTS

In this section, we present and compare the results using the proposed architecture against other methods in the literature. We present the performance of our architecture using the RMSE and scoring function. We also present the attention activations to visualise time-related features used for RUL prediction at each time step.

5.1. LSTM Performance

We implement two versions of the proposed architecture: one containing the Attention layer (LSTM + A) and one without Attention mechanism (LSTM + FFNN). We also present a comparison of the models to the *state-of-the-art* results in the C-MAPPS datasets for the RMSE and Scoring performance metrics in Tables 3 and 4.

We compare our model to other Deep Learning architectures applied to the same datasets. We compare to LSTM methods proposed by Listou Ellefsen et al. (2019) (GA + LSTM) and Zheng et al. (2017) as LSTM + FFNN as to test whether our LSTM implementation can offer any gains over previously implemented LSTM architectures. Our approach is similar to the one proposed by Zheng et al. (2017), but here we do not read an entire sequence of inputs to make an RUL esti-

	Scoring Function				
Method	FD001	FD002	FD003	FD004	R_e
LSTM + FFNN (Zheng et al., 2017)	338	4,450	852	5,550	130
MODBNE (Zhang et al., 2017)	334	5,585	422	6,558	-
CNN + FFNN (Li et al., 2018)	274	10,412	284	12,466	125
GA + LSTM (Listou Ellefsen et al., 2019)	**231**	3,366	251	**2,840**	115-135
Proposed LSTM + FFNN ($\pm$ StDev)	300 ($\pm$ 31)	**1,638** ($\pm$ 192)	267 ($\pm$ 42)	2,904 ($\pm$ 979)	125
Proposed LSTM + A ($\pm$ StDev)	320 ($\pm$ 30)	2,102 ($\pm$ 250)	**223** ($\pm$ 17)	3,100 ($\pm$ 576)	125

Table 4. Scoring function comparison between the proposed LSTM methods and other methods in the literature on the C-MAPPS datasets

mation for each time step in the input sequence, i.e. while predicting for time the next time step $t + 1$ we learn hidden vectors over a sequence size T_w until time t. Moreover, different from the previously proposed LSTM model we do not perform pre-clustering of operating conditions in datasets FD002 and FD004 nor perform unsupervised pretraining as reported in Listou Ellefsen et al. (2019). We also compare to the Convolutional Neural Network (CNN + FFNN) methodology proposed by Li et al. (2018) and the Multiobjective Deep Belief Networks Ensemble (MODBNE) proposed by Zhang et al. (2017) which presented high-performance results in the datasets.

In Table 3, we present the performance results of the proposed model using the RMSE performance function. As it can be seen from the results, our model achieves lower RMSE values for datasets FD002 and FD004. Thus, our proposed method can yield better performance on datasets with more operating conditions. Our model results in 21% (FD002) and 11% (FD004) relative improvement over the previously reported RMSE. For the remaining datasets, our models are comparable in performance with the ones reported by Listou Ellefsen et al. (2019) (GA + LSTM), resulting in 8% (FD001) and 3% (FD003) performance reduction. We point out that unlike the GA + LSTM method we do not use heuristic search to select the best performing hyperparameters or incur in unsupervised pretraining of the network weights.

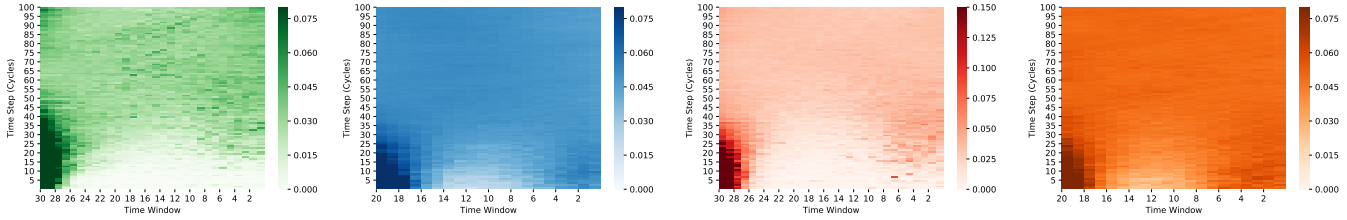
We compare our models in more detail with LSTM + FFNN and GA + LSTM. In the first, the architecture proposed is similar to our model, however, several differences are present. In our implementation, we do not pre-cluster the operating conditions on datasets FD002 and FD004. Moreover, we use a ReLU activation function in the network, which matches the input range of our normalised sensors. Moreover, as seen in Section 4.4, tuning the time-window size leads to the most improvement in our experiments. When compared to GA + LSTM, our method differs on the lack of pretraining of the network and in the overall final architecture, including activation functions. Furthermore, we use a much smaller time window for training. Lastly, different from both methods, we normalise both inputs and outputs to the 0-1 range, which helps to stabilise learning. After weight optimisation, the output is multiplied by R_e to recover the original (rectified) values and compared to RUL values from the test datasets also rectified by R_e. As shown in Tables 3 and 4, the rectified RUL values are similar to other others in the literature and match the ones proposed in Li et al. (2018).

The modifications on the architecture result in higher performance gains for FD002 and FD004 in comparison to the other datasets. In our experiments, we have noted that including the raw operating conditions features in the model, selecting the correct time-window for propagating gradients in the network and increasing the number of hidden neurons in the LSTM layers led to the most benefits for these datasets. Different from previous approaches, using non-clustered operational settings leads to better predictions in our proposed architecture. Moreover, increasing the number of hidden neurons increases the capacity of the Neural Network of extracting more complex features found in the input sequences of FD002 and FD004. We argue that this modification also improves performance in the attentional model as the attention mechanism can attend to a larger and more diverse hidden representation of the input space before a prediction.

Similarly, in Table 4, our methods can achieve much lower scoring values for dataset FD002 yielding a 52% relative improvement over the best-reported results. Here, the benefits come from the network being able to reduce late RUL predictions. In our tests, the average observed rectified RUL of FD002 is 73.69 while the predictions coming from our method average at 71.34. These results do not necessarily translate in the same magnitude to the RMSE results as the RMSE metric weighs both positive and negative errors equally. As presented in Section 4.4, the choice of hyperparameters, in particular, the time window size and number of neurons in the LSTM layers translate to the highest performance gain in our method. For the remaining datasets, our methods present similar performance to previously proposed methods.

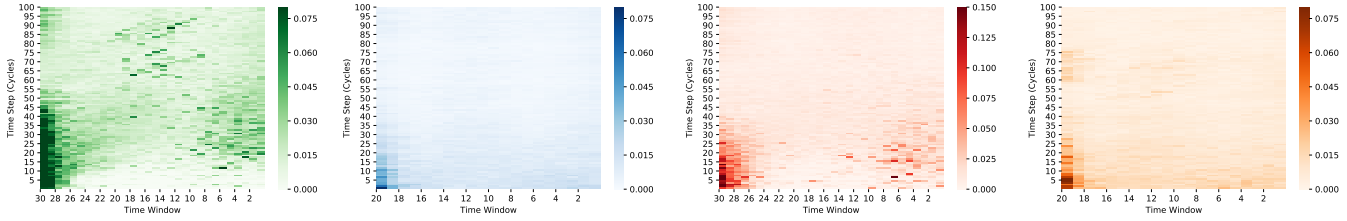
5.2. Attention Weights

In our experiments, the model combined with an attention layer has achieved similar performance to the models contain-



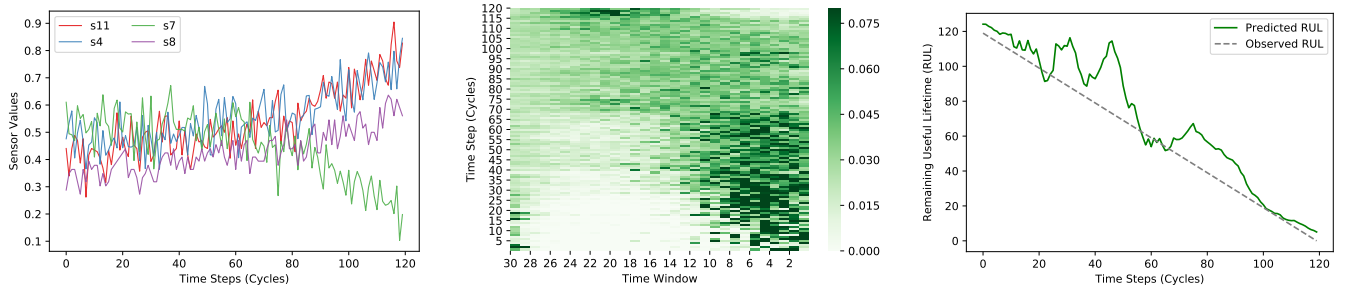
(a) FD001 (b) FD002 (c) FD003 (d) FD004

Figure 8. Average attention weights starting 100 time steps before a failure for cross-validation examples in each dataset.



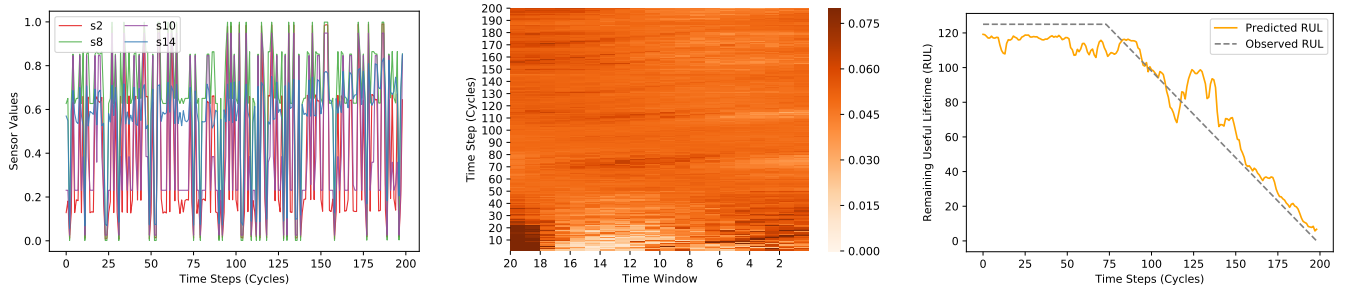
(a) FD001 (b) FD002 (c) FD003 (d) FD004

Figure 9. Standard deviation of attention weights starting 100 time steps before a failure for cross-validation examples in each dataset.



(a) Sensor values over time. (b) Attention weights at each time step. (c) Predicted RUL using LSTM + A and Observed RUL targets.

Figure 10. Sensor values, attention weights and Remaining Useful Lifetime predictions for example ID 8 in FD001.



(a) Sensor values over time. (b) Attention weights at each time step. (c) Predicted RUL using LSTM + A and Observed RUL targets.

Figure 11. Sensor values, attention weights and Remaining Useful Lifetime predictions for example ID 32 in FD004.

ing just LSTM and Fully-Connected layers. A small improvement in RMSE can be achieved for FD004 and FD002. While for the scoring function, the LSTM + A model only outperforms the original LSTM model for FD003 dataset. However, the attention weights in Eq. (9) can be retrieved to look at the importance the networks give to each time step of the context vector. This can help us interpret which timestep the network focus on to make the RUL predictions for the next time step.

It is important to notice that in our attention mechanism the network has access to both a context vector and the last hidden state h_t. This architecture choice has the effect of forcing the network to attend to parts different from the last time step before a prediction. Which we expect to be the most relevant for the prediction at the next time step. As we present below, this has a direct impact on the learned attention weights in our experiments.

In Figures 8 and 9, we present the average and standard deviations of attention weights over cross-validation examples starting 100 time steps before a failure. In the figures, the vertical axis corresponds to a time step and the horizontal axis represents the size of the sliding window, with the leftmost side representing the farthest time step from the current RUL prediction i.e. $x^i_{t-T_w+1}$. In the figures, we observe that at the start of the predictions the networks does not focus on specific parts of the inputs to make a prediction. However, as time progresses and the predictions approach the end of lifetime attention is shifted to the either the first few time steps of the prediction window or to the last timesteps (closer to the prediction point). This is can be seen in more details for FD001, in Figures 8(a) and 9(a), and for FD003 in Figures 8(c) and 9(c). On average, at the end of the lifetime, the attention shifts towards the start of the time window but the standard deviation values show that attention is also present at the end of the sliding window due to their high values.

We also present two specific examples, one for FD001 in Figure 10 and one for FD004 in Figure 11. In the figures, we present selected sensor values, the learned attention weights and the RUL prediction at each time step for an example coming from the respective datasets. In Figure 10, we observe how attention weights shift towards the end of the time window as sensors start to show a degradation trend. This indicates that the network can attend to important parts of the input features for the RUL prediction. We also notice that the learned attention weights are lower in the central part of the time window as degradation occurs. As the trend and slope of the curves are important for degradation estimation, the network could have learned that as time passes it needs to focus on how much the curve is changing in the given time window. In Figure 11, the sensors in Figure 11(a) present a more erratic behaviour and sensor tend to show small changes of slope over time. This effect is due to different operating conditions in the given dataset. More importantly, the learned attention weights in Figure 11(b) show that the network has learned to focus its attention on similar parts of the input as time progresses, only focusing at the beginning and end of the time window as RUL approaches zero.

These results offer new insights as of how the time-related features are used by the LSTM architecture while making RUL predictions. For example, a network containing attention mechanisms trained to identify faulty behaviour can be used to visually inspect input sensor values as time progresses. As it is hard to visually inspect all incoming sensor data, attention mechanisms could offer a visualisation method for fault prognostics and identification. In such cases, a temporal visual inspection would offer a visual representation as to when faulty behaviour starts. Such, early warnings could be used to present future failures in complex systems.

6. CONCLUSION

In this work, we proposed an LSTM architecture for RUL prediction of turbofan degradation engines using the C-MAPPS datasets. We proposed an architecture containing an attention mechanism that has been used to visualise the temporal relationships between inputs and predicted RUL outputs.

Our results show that the proposed methodology is competitive with other proposed methods in RUL predictions. We show the effectiveness of the model in comparison to other Deep Learning methods previously proposed for the same data. Our attention weights show that the LSTM network focuses on different parts of the temporal input while making a prediction depending on which part of the degradation cycle considered. Results of the attention mechanism can be used for better interpretability of Deep Learning approaches.

As limitations to the proposed method, we point out that our approach does not offer temporal attention relationships for each input variable separately. Such an approach could be used to visually identify specific faulty components before failure occurs. Moreover, other empirical results in different PHM datasets are necessary to validate the methodology to different use cases. Lastly, our attention mechanism works over latent temporal features of the inputs, an extension could incorporate other architectures that incorporate temporal self-attention mechanisms while requiring less computational power than LSTM networks.

ACKNOWLEDGMENTS

This work was supported by the "Netherlands Organisation for Scientific Research" (NWO). Project: NWO Big data - Real Time ICT for Logistics. Number: 628.009.012

REFERENCES

Abadi, M., et al. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/`

Babu, G. S., Zhao, P., & Li, X.-L. (2016). Deep convolutional neural network based regression approach for estimation of remaining useful life. In *International Conference on Database Systems for Advanced Applications* (pp. 214–228).

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine

translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015.*

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Benkedjouh, T., Medjaher, K., Zerhouni, N., & Rechak, S. (2013). Remaining useful life estimation based on nonlinear feature reduction and support vector regression. *Engineering Applications of Artificial Intelligence*, *26*(7), 1751–1760.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1724–1734).

Chollet, F., et al. (2015). *Keras.* Retrieved from `https://keras.io`

da Costa, P. R. d. O., Akçay, A., Zhang, Y., & Kaymak, U. (2020). Remaining useful lifetime prediction via deep domain adaptation. *Reliability Engineering & System Safety*, *195*, 106682.

Galassi, A., Lippi, M., & Torroni, P. (2019). Attention, please! a critical review of neural attention models in natural language processing. *arXiv preprint arXiv:1902.02181*.

He, D., & Bechhoefer, E. (2008). Development and validation of bearing diagnostic and prognostic tools using hums condition indicators. In *2008 IEEE Aerospace Conference* (p. 1-8).

Heimes, F. O. (2008). Recurrent neural networks for remaining useful life estimation. In *2008 International Conference on Prognostics and Health Management* (p. 1-6).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hossain, M., Sohel, F., Shiratuddin, M. F., & Laga, H. (2019). A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys (CSUR)*, *51*(6), 118.

Huang, R., Xi, L., Li, X., Liu, C. R., Qiu, H., & Lee, J. (2007). Residual life predictions for ball bearings based on self-organizing map and back propagation neural network methods. *Mechanical Systems and Signal Processing*, *21*(1), 193–207.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015.*

Lei, Y., Li, N., Guo, L., Li, N., Yan, T., & Lin, J. (2018). Machinery health prognostics: A systematic review from data acquisition to rul prediction. *Mechanical Systems and Signal Processing*, *104*, 799–834.

Li, X., Ding, Q., & Sun, J. Q. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering and System Safety*, *172*, 1–11.

Listou Ellefsen, A., Bjørlykhaug, E., Æsøy, V., Ushakov, S., & Zhang, H. (2019). Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture. *Reliability Engineering & System Safety*, *183*, 240–251.

Luong, T., Pham, H., & Manning, C. D. (2015, September). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1412–1421). Association for Computational Linguistics.

Papakostas, N., Papachatzakis, P., Xanthakis, V., Mourtzis, D., & Chryssolouris, G. (2010). An approach to operational aircraft maintenance planning. *Decision Support Systems*, *48*(4), 604–612.

Saxena, A., Goebel, K., Simon, D., & Eklund, N. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 International Conference on Prognostics and Health Management, PHM 2008* (pp. 1–9).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Tian, Z. (2012). An artificial neural network method for remaining useful life prediction of equipment subject to condition monitoring. *Journal of Intelligent Manufacturing*, *23*(2), 227–237.

Wu, Y., Yuan, M., Dong, S., Lin, L., & Liu, Y. (2018). Neurocomputing Remaining useful life estimation of engineered systems using vanilla LSTM neural networks. *Neurocomputing*, *275*, 167–179.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning* (pp. 2048–2057).

Yuan, M., Wu, Y., & Lin, L. (2016). Fault diagnosis and remaining useful life estimation of aero engine using lstm neural network. In *2016 IEEE International Conference on Aircraft Utility Systems (AUS)* (p. 135-140).

Zhang, C., Lim, P., Qin, A., & Tan, K. C. (2017). Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics. *IEEE Transactions on Neural Networks and Learning systems*, *28*(10), 2306–2318.

Zheng, S., Ristovski, K., Farahat, A., & Gupta, C. (2017). Long Short-Term Memory Network for Remaining Useful

Life estimation. In *2017 IEEE International Conference on Prognostics and Health Management, ICPHM 2017* (pp. 88–95).

Zio, E., & Di Maio, F. (2010). A data-driven fuzzy approach for predicting the remaining useful life in dynamic failure scenarios of a nuclear system. *Reliability Engineering & System Safety*, *95*(1), 49–57.