Uncertainty Estimation in RUL prediction task

Serop Baghdadlian
Friedrich-Alexander-University Erlangen-Nuremberg
Erlangen, Germany
serop.baghdadlian@fau.de

***Abstract*—In this paper, the application of uncertainty estimation in a Remaining useful life (RUL) prediction task is examined. Implementing AI solutions in high risk application such as autonomous driving, medicine and rul estimation requires the knowledge of the certainty of the AI model. This later aims to incorporate human intervention when uncertainty increases greatly and an automatic decision is hard to make due to noise or unseen data. Several methods to estimate uncertainty measures have been researched, however applying these methods on RUL estimation task was not yet researched. Therefore this paper provides an overview of the application of Dropout inference and Bayesian Neural networks to capture prediction uncertainty in the RUL estimation. These techniques were used alongside Long short term memory model to predict the RUL of turbo fan engines simulations generated by the C-MAPS dataset. The results of this paper suggest that dropout inference is a fast and efficient methods to provide uncertainty estimation. This uncertainty also increased increased upon the addition of random Gaussian noise to the dataset. Bayesian LSTM, on the other hand, was less accurate in modelling the RUL curve and less sensitive to noise addition.**

***Index Terms*—Uncertainty, RUL, LSTM, Bayesian network, Deep learning**

I. Introduction

The increased economic challenges have lead companies to invest a lot of time in maximising their profits. This was possible either by increasing their price and productivity or by minimising their expenses. One of the biggest expenses manufacturers face are maintenance costs, which accounted for almost 15% - 70% of their total expenses [2]. Consequently, this has highlighted the importance of analytical methods that can proactively predict when future repair or replacement are necessary. These techniques are referred to as condition-based maintenance (CBM) [12]. On the other hand, predicting the health status of the machine is referred to as remaining useful life (RUL) [12] and is defined as the length of the period between the current time and the time where the object is no longer useful i.e. defect [17].

Data based approaches to estimate the RUL has been widely adopted in research. These methods provide predictions using the original input data without prior domain knowledge or feature engineering [17]. The research interest in this area has increased greatly for the last 15 years and one of the most successful techniques currently used are deep learning techniques. Long short term memory (LSTM), is a deep learning architecture widely used in temporal data, such as sensor data, and has shown good abilities to learn complex temporal representations and patterns [2]. It has been specially deployed to solve the RUL task and achieved very good results [18].

On the other hand, deep learning and neural network approach show some weak spots. They provide high confidence outputs despite being shown noisy inputs or inputs from outside their training distribution [10][13]. This problem created the need for uncertainty measures that represents the networks confidence when faced with unseen or noisy data. This could provide a window for human intervention in high risks domain, such as autonomous driving and RUL estimation that prevent human and financial losses [5].

Many methods for the uncertainty estimation have been introduced, such as the deep ensemble [9], dropout inference [6] and lately Bayesian neural networks [10]. While almost all of these techniques rely on the idea of training many different models and averaging their decisions, they differ in the way they achieve this variance. For a extensively summary of current state of uncertainty in deep learning please refer to [5].

While many techniques to perform both RUL estimation and uncertainty estimation separately exist, the literature was lacking successful application of uncertainty measures in context of RUL estimation task. Thus, the research question of this paper is: How can uncertainty estimation be performed on RUL prediction tasks and how does this uncertainty behave in the presence of input noise?

The remainder of this paper is organised as follows: Section II describes the methods within the scope of this paper, including the used dataset, the implemented deep learning model and which uncertainty measures were applied. Section III displays the results obtained from extensive experiments on the data. Section IV provides a discussion of these results and answers the research questions and finally Section V provides a conclusion of the findings of this paper and possible future improvements.

II. Methods

A. Data

The data used within this paper is a simulated turbo fan engine dataset from NASA. This data is collected from the commercial software C-MAPS, which creates a simulated physical model of the turbo fan engine and can induce

artificial faults that simulates a real world technical failure [15]. During the simulation period, readings from 24 different sensors are recorded until the point of total engine failure. As a result, the acquired data is therefore time series data, where a time point within the life cycle of the engine is referred to a cycle. The Data contains 4 training and testing datasets, where each dataset has a different number of simulated engines, differing in the starting values and fault initiation time.

Before training, sensor values were normalised using the standard scalar according to:

$$Z = (x - \mu)/\sigma \tag{1}$$

Where μ and σ are the mean and variance sensor value, respectively. Z is the new normalised sensor value.

In the original dataset, the total number of cycles (max RUL) was given. To calculate the RUL value at each cycle of the engine, some preprocessing steps was necessary. First, the value for RUL_{early}, which represent a constant showing the maximum number of cycles that an engine is expected to last, was set to 125 cycles according to literature comparison done in [11]. From this value the $Knee_{point}$ was then calculated, which represents the point in time where the engine's health starts deteriorating. It is calculated by subtracting the maximum number of cycles from the RUL_{early} [8]. After acquiring these values, the time sequences for each engine are assigned their true label RUL value according to the following:

$$Rul_t = \begin{cases} Rul_e & \text{if } cycle \leq Knee_{pnt} \\ Rul_{t-1} - \frac{Rul_e}{Max_c - Knee_{pnt}} & \text{else} \end{cases},$$

Where, RUL_t represents the RUL value at cycle t in the sequence, RUL_e is the RUL_{early}.

Finally to use the data for training and testing a rolling window approach was used with a window of 30 cycles [11]. The window is shifted by 1 cycle to capture as much data as possible. Values in the beginning of the sequence were padded with zeros in order to be included in the training as well. Figure 1 describes the process windowing approach used in this paper.

B. Models

To perform the RUL prediction, a LSTM deep learning model was used due to its ability to learn time series patterns. It also showed superior results on this dataset compared to other methods such as CNN and SVM techniques [18]. One of the main advantages of deep learning methods is the ability to use original data as a direct input without the need for feature engineering. The architecture used contained two LSTM layers with 32, 64 neurons, respectively, followed by two fully connected layers with 8 neurons each. This configuration was chosen according to a recommendation in [18].

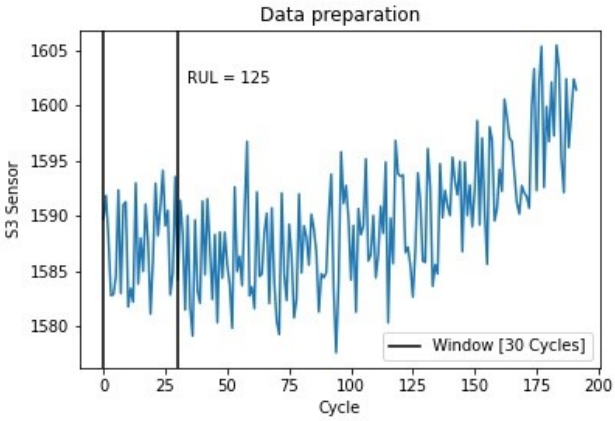

Fig. 1: Rolling window method

To model the uncertainty two approaches were used:

1) Bayesian LSTM: As mentioned in I, Bayesian inference is one of the most widely used methods to include uncertainty in neural network's decision. It is based on Bayesian neural networks, which trains and optimise a probability distribution rather than a deterministic set of weights. To add uncertainty, both the weights and bias are sampled before the feed forward process according to the following:

$$W_n^i = \mathcal{N}(0,\, 1) * \log\left(1 + \rho^i\right) + \mu^i. \tag{2}$$

$$b_n^i = \mathcal{N}(0,\, 1) * \log\left(1 + \rho^i\right) + \mu^i. \tag{3}$$

Where $\mathcal{N}(0,\, 1)$ is a Gaussian distribution with zero mean and unit variance. In these equations the trainable parameters are μ and ρ which parameterize the distribution of the weights and therefore are updated during training.

The Goal of the Bayesian learning to find a suitable posterior distributions for the weights that best explains the dataset. This is normally achieved by maximizing the posterior probability with respect to the weights according to the following:

$$W_{MAP} = arg\max_{w} log\, P(w/D) \tag{4}$$

However, Solving this posterior probability is analytically infeasible, therefore the literature tends to solve it by approximating a distribution $q(w/\theta)$, which is close to it, by minimising the Kullback Lieber divergence according to parameter vector θ [7]. According to [4] this minimisation is equal to the minimisation of the quantity termed variational free energy according to the following:

$$F(D, \theta) = KL[q(w/\theta)//P(W)] - E_{q(w/\theta)}[logP(D/w] \tag{5}$$

This is however also analytically infeasible. Finally, after many research approaches this cost function can be approximated using stochastic variational inference and Monte Carlo sampling according to the following:

$$F(D,\theta) \approx \sum_i logq(w_i/\theta) - logP(w_i) - logP(D/w_i) \quad (6)$$

From 6 the updates for the μ and ρ parameters can then be calculated using the gradient of the cost function $f(w,\theta)$ with respect to μ and ρ plus the gradient with respect to the weights w.

2) Dropout: Dropout is a regularisation technique that helps neural networks against over-fitting the training data. Over-fitting is term given to neural network when they provide a costume solution that works only on training data, but fails to generalise on other similar test data [16]. This is achieved by randomly setting the output of neural networks to zero. This forces the network not to overweight specific paths within the network and ultimately increase its generalisation power. This property could also be used during inference so that the input of the network will be multiplied with different set of weights and biases within the network and therefore reach different outputs each time it is propagated within the network. Propagating the same input multiple times will result in different outputs. The mean of these outputs is then considered as the new output of the network and the variance between these outputs is seen as uncertainty measure. This technique resembles an ensemble of decisions made by many smaller networks and the variance in the decisions was used as uncertainty measure.

C. Software and Hardware

The software used to do the experiments is written in the programming language python and used the framework Pytorch [14] for LSTM implementation. The library Blitz was used to implement the Bayesian LSTM, it contains necessary tools and functions to perform Bayesian inference and variational Bayesian learning [3]. For training, Google Collab was used with a GPU (Nvidia P100) with 12.72 GB RAM.

D. Training

For training, the chosen batch size was 512, a value recommended in literature [11]. The learning rate was set to 0.001 and decayed by 0.1 after specific number of epochs. The normal LSTM was trained for 60 Epochs while the Bayesian LSTM only for 5 Epochs. This was due to the large memory demand from the Bayesian LSTM, which led to an out of memory error after 5 Epochs. Furthermore, the training dataset was separated in training and validation sets with 80:20 ratio, respectively. As a loss function the root mean squared error (RMSE) was most commonly used in related literature and therefore was also applied in the training [11][18].

E. Testing

C-Maps dataset contains 4 different testing sets that has slight different structure than training data. Here the sensor recordings were stopped before total failure of the engine and RUL value at the last recorded cycle is then given as the true label for the engine's recordings. In order to compare the acquired results with literature references, RMSE was used to test the accuracy of the trained LSTM models. To test the uncertainty properties even further, random Gaussian noise, with zero mean and standard deviation of 0.1 was added to the test data at four specific points 20, 50, 70 and 99% for the max cycle of the engine. For example if the maximum cycle of an engine is 450 cycles, the first random noise is applied at cycle number 90, and so on.

III. Results

During training both normal and Bayesian LSTM showed a good decrease in the loss function, however, the Bayesian LSTM did over-fit the training data already after only 5 epochs. After training, the normal LSTM model and dropout inference achieved comparable mean loss of 11.77 on the 4 test dataset. This value is much better than the stated value of 21.25 in [18]. Bayesian LSTM on the other hand achieved a higher loss value 16.12. Adding random noise has also resulted in addition increase of circa 0.2 for both the normal and Bayesian LSTM.

The results of a randomly chosen sample of the test data is shown in the figures from 2 to 10. Figures 2 and 3 show the results of using normal LSTM on the test data. Figures 5 and 6 show the results of applying dropout inference during prediction, while figures 8 and 9 show the results of Bayesian LSTM. In the dropout and Bayesian inference the uncertainty is represented in red, the predicted mean value in blue, while the true label is shown in black.

To get uncertainty measures for the test data, the same input was propagated into the network five time and mean value was taken as the output of the network while the standard deviation was represented as the uncertainty measure.

Figure 4 shows the good RUL estimation power of the LSTM. The curve is model accurately with a slight shift. Furthermore, adding dropout inference to the LSTM shows that the true label was within the uncertainty interval more than 90% of the time as shown in figure 5. Bayesian LSTM on the other hand, shows less accurate modelling of the RUL curve where the true label was more than 70% outside the uncertainty interval.

Looking at the curves with noise addition, a slight drops in the predicted values for the normal LSTM can be noticed at the four positions where the noise is added. This has, on the other hand, resulted in an increase in the uncertainty interval using dropout inference at these specific positions. The same effect was, however, less obvious using the Bayesian LSTM

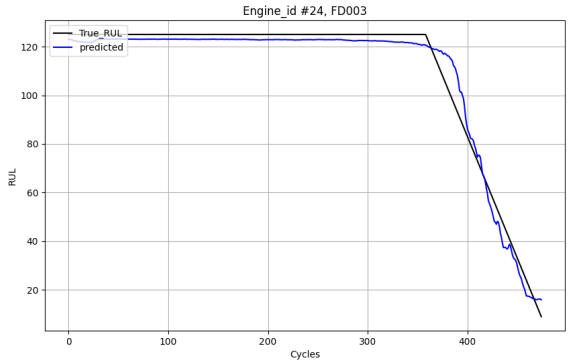

Fig. 2: Without noise addition

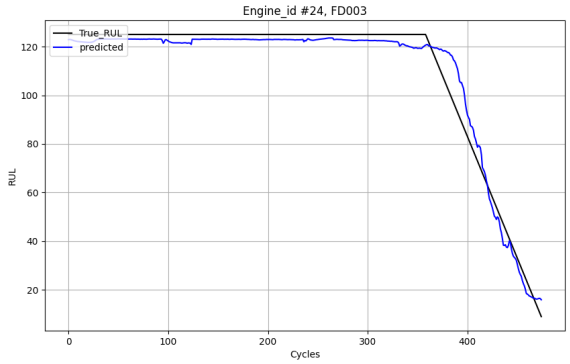

Fig. 3: With noise addition

Fig. 4: RUL prediction of engine Nr. 24 using normal LSTM

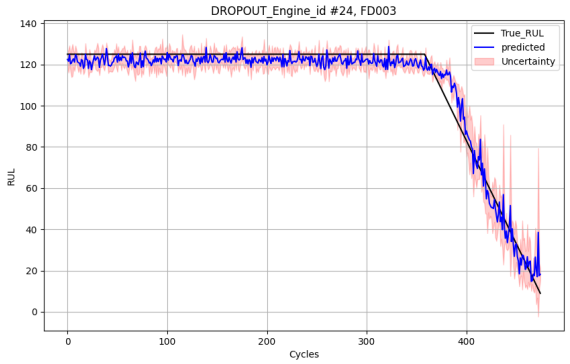

Fig. 5: Without noise addition

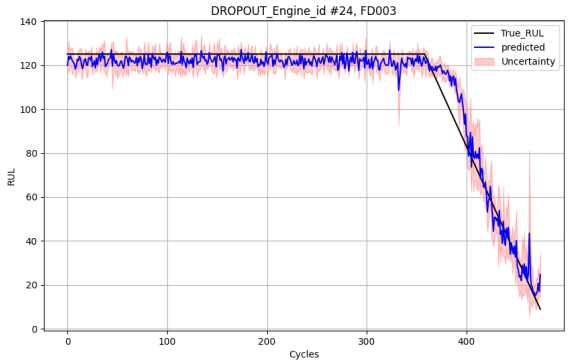

Fig. 6: With noise addition

Fig. 7: Rul prediction of engine Nr. 24 using Dropout inference

as shown in figure 9, where the mean predicted values has also dropped greatly with this addition and the uncertainty interval stayed almost the same as without noise.

IV. Discussion

As seen in the section III, using LSTM layers to approximate the RUL curve is a good option and achieved quite high accuracy. It requires no domain knowledge and could be implemented and deployed quite easily with the help of frameworks such as Tensorflow and Pytorch.

Adding dropout inference to the LSTM architecture produced a nice and accurate uncertainty measure which increased at positions, where the Gaussian noise was added. This was achieved with the same speed and memory demand as the normal LSTM. This behaviour was consistent within the majority of the test data and therefore could be used a reliable uncertainty measure for the RUL estimation task.

Bayesian inference, on the other hands, was much slower to train and during deployment. It required much more memory demand till the point of out of memory error mentioned in section II. However, since the purpose of this paper is to compare different uncertainty techniques, the architecture used for both models was kept the same. On the test data, Bayesian inference has unperformed and didn't produce as accurate RUL prediction such as the one shown in figure 5. The predicted values shown in figure 8 were shifted by almost 20 cycles, but eventually the predicted RUL were close to the true labelled one. It should be noted that also the positions, where the noise was added, has enquired a huge drop in the RUL value without an increase in the uncertainty interval. Only the noise added to the end of the measurement sequence induced an increase in the uncertainty without a sudden jump in the mean value.

This behaviour was quite consistent within the entire testing dataset. This could be due to the over-fitting problem and insufficient training and therefore could be improved by experimenting with different architecture designs and costume training. Moreover, as mentioned in subsection II-D, the Bayesian LSTM model was trained only to 5 epochs due to out of memory errors. During this 5 epochs however, the model over-fit the training dataset due to the huge number of parameters used by the Bayesian model. Interesting to observe was that the number of parameters kept increasing after each epoch which lead to an out of memory error. For

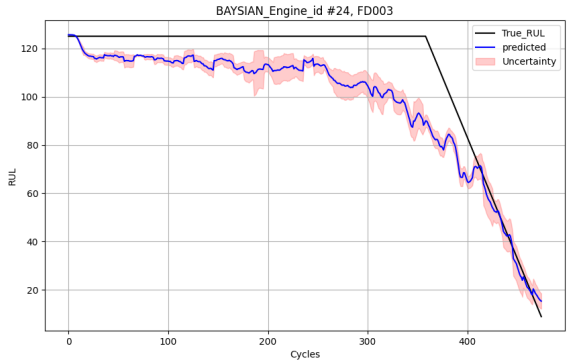

Fig. 8: Without noise addition

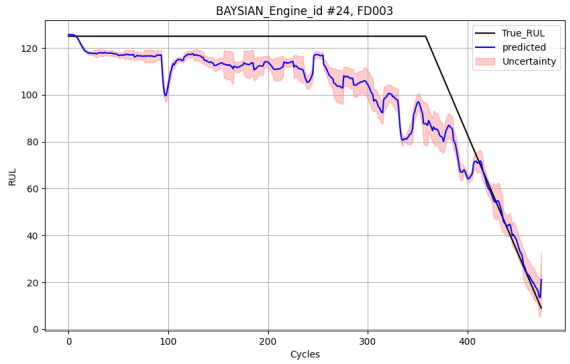

Fig. 9: With noise addition

Fig. 10: Rul prediction of engine Nr. 24 using Bayesian inference

this a ticket was issued to the Github repository of the library Blizt and can be found under [1].

Finally, adding uncertainty measures has proven to be a quite important technique and a great addition to the RUL prediction tasks, where more predictable and explainable neural network outputs could lead to a better prediction and higher deployment rate even within high risk tasks such as the RUL task. For this purpose, applying efficient and fast solution to get uncertainty such as by using dropout inference can be recommended.

V. Conclusion

The purpose of this paper was to estimate uncertainty and its impact on a relatively hard task such as RUL prediction. For this purpose two different uncertainty estimation techniques were used and compared with each. To perform the RUL prediction a LSTM model was used. This LSTM model was combined with dropout and Bayesian inference to get an uncertainty measures to its outputs. The C-MAPS dataset from NASA was used to perform this task.

Dropout inference on the LSTM model achieved the highest accuracy during training and testing. It also showed the most reliable uncertainty estimation measure, where uncertainty has increased greatly after adding random noise to the input data. Bayesian LSTM of the same architecture has resulted in over-fitting the training data and performing poorly on the test dataset. It has, however, increased its uncertainty interval at only one from 4 positions, where the noise was added. In the other three positions the mean predicted value also dropped significantly.

This shows that uncertainty measures in RUL prediction task can be very helpful in distinguishing noisy data and provide a reliability measure for neural network's output, especially in a high risk task such as RUL prediction. Also, for efficiency, dropout inference has shown to be the most efficient and intuitive solution to achieve uncertainty measures, while Bayesian model required costume training and increased computational power.

While the findings of this paper has shown poor Bayesian LSTM results, costume architecture design and longer training could improve the results and achieve better RUL and uncertainty modelling. For this, we advise to use smaller Bayesian models than the one used within this paper and train on GPUs with larger capacities than the one in Google Collab. It could also be interesting to use other frameworks to implement the Bayesian LSTM rather than Pytorch and Blizt. There exist already finished implementations using Tensorflow for the Bayesian LSTM that could be more efficient. Also it would be interesting to customise a loss function that uses the generated uncertainty interval for training, for example, penalising small uncertainty for noisy data. Finally, while this paper has only compared two uncertainty measures, other techniques could be more suitable for this specific task and therefore could be important to make a systematic comparison between these techniques and the task of RUL prediction.

References

[1] serop baghdadlian. *Huge Memory Demand*. URL: https://github.com/piEsposito/blitz-bayesian-deep-learning/issues/68.

[2] Maurizio Bevilacqua and Marcello Braglia. "Analytic hierarchy process applied to maintenance strategy selection". In: *Reliability Engineering System Safety* 70 (Oct. 2000), pp. 71–83. DOI: 10.1016/S0951-8320(00)00047-8.

[3] Piero Esposito. *BLiTZ - Bayesian Layers in Torch Zoo (a Bayesian Deep Learing library for Torch)*. https://github.com/piEsposito/blitz-bayesian-deep-learning/. 2020.

[4] Karl Friston et al. "Variational free energy and the Laplace approximation". English. In: *NeuroImage* 34.1 (Jan. 2007), pp. 220–234. ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2006.08.035.

[5] Yarin Gal. “Uncertainty in Deep Learning”. PhD thesis. University of Cambridge, 2016.

[6] Yarin Gal and Zoubin Ghahramani. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1050–1059. URL: http://proceedings.mlr.press/v48/gal16.html.

[7] Geoffrey E. Hinton and Drew van Camp. “Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights”. In: *Proceedings of the Sixth Annual Conference on Computational Learning Theory*. COLT ’93. Santa Cruz, California, USA: Association for Computing Machinery, 1993, pp. 5–13. ISBN: 0897916115. DOI: 10.1145/168304.168306. URL: https://doi.org/10.1145/168304.168306.

[8] LahiruJayasinghe. *RUL-Net*. 2018. URL: https://github.com/LahiruJayasinghe/RUL-Net.

[9] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. “Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles”. In: (Dec. 2016).

[10] Jongseok Lee et al. “Estimating Model Uncertainty of Neural Networks in Sparse Information Form”. In: July 2020.

[11] Xiang Li, Qian Ding, and J. Q. Sun. “Remaining Useful Life Estimation in Prognostics Using Deep Convolution Neural Networks”. In: *Reliability Engineering System Safety* 172 (Dec. 2017). DOI: 10.1016/j.ress.2017.11.021.

[12] Xiang Li, Qian Ding, and Jian-Qiao Sun. “Remaining useful life estimation in prognostics using deep convolution neural networks”. In: *Reliability Engineering System Safety* 172 (2018), pp. 1–11. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2017.11.021. URL: https://www.sciencedirect.com/science/article/pii/S0951832017307779.

[13] A. Loquercio, M. Segu, and D. Scaramuzza. “A General Framework for Uncertainty Estimation in Deep Learning”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3153–3160. DOI: 10.1109/LRA.2020.2974682.

[14] *Pytorch*. URL: https://pytorch.org/.

[15] A. Saxena and K. Goebe. *Turbofan Engine Degradation Simulation Data Set”, NASA Ames Prognostics Data Repository*. 2008. URL: http://ti.arc.nasa.gov/project/prognostic-data-repository.

[16] Wikipedia. *Overfitting*. 2017. URL: https://en.wikipedia.org/wiki/Overfitting.

[17] S. Zheng et al. “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. DOI: 10.1109/ICPHM.2017.7998311.

[18] S. Zheng et al. “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. DOI: 10.1109/ICPHM.2017.7998311.