A Neural Network-Evolutionary Computational Framework for Remaining Useful Life Estimation of Mechanical Systems

David Laredo[1], Zhaoyin Chen, Oliver Schütze[2] and J. Q. Sun, Jingwen Huang[1],Tingting Zhang[1] and Jian-Qiao Sun[2]

[1]*Beijing University of Chemical Technology*
Beijing, 100029, China
[2]*Department of Mechanical Engineering*
School of Engineering, University of California
Merced, CA 95343, USA
Corresponding author. Email: jqsun@ucmerced.edu

Abstract

This paper presents a framework for estimating the remaining useful life (RUL) of mechanical systems. The framework consists of a multi-layer perceptron and an evolutionary algorithm for optimizing the data-related parameters. The framework makes use of a strided time window along with a piecewise linear model to estimate the RUL for each time window in the training set. Tuning the data-related parameters in the optimization framework allows for the use of simple models, e.g. neural networks with few hidden layers and few neurons at each layer, which may be deployed in environments with limited resources such as embedded systems. The proposed method is evaluated on the publicly available CMAPS dataset. The accuracy of the proposed method is compared against other state-of-the art methods in the literature. The proposed method is shown to perform better while making use of compact model.

Keywords: artificial neural networks, moving time window, RUL estimation, prognostics, evolutionary algorithms

1. Introduction

Traditionally, maintenance of mechanical systems has been carried out based on scheduling strategies. Such strategies are often costly and less capable of meeting the increasing demand of efficiency and reliability [1, 2]. Condition based maintenance (CBM) also known as intelligent prognostics and health management (PHM) allows

for maintenance based on the current health of the system, thus cutting down the costs and increasing the reliability of the system [3]. Here, we refer to prognostics as the estimation of remaining useful life of a system. The remaining useful life (RUL) of the system can be estimated based on the historical data. This data-driven approach can help optimize maintenance schedules to avoid engineering failures and to save the costs [4].

The existing PHM methods can be grouped into three different categories: model-based [5], data-driven [6, 7] and hybrid approaches [8, 9]. Model-based approaches attempt to incorporate physical models of the system into the estimation of the RUL. If the system degradation is modeled precisely, model-based approaches usually exhibit better performance than data-driven approaches [10]. This comes at the expense of having extensive a priori knowledge of the underlying system and having a fine-grained model of the system, which can involve expensive computations. On the other hand, data-driven approaches use pattern recognition to detect changes in system states. Data-driven approaches are appropriate when the understanding of the first principles of the system dynamics is not comprehensive or when the system is sufficiently complex such as jet engines, car engines and complex machineries, for which it is prohibitively difficult to develop an accurate model.

Common disadvantages for the data-driven approaches are that they usually exhibit wider confidence intervals than model-based approaches and that a fair amount of data is required for training. Many data-driven algorithms have been proposed. Good prognostics results have been achieved. Among the most popular algorithms we can find artificial neural networks (ANNs) [11], support vector machine (SVM) [12], Markov hidden chains (MHC) [13] and so on. Over the past few years, data-driven approaches have gained more attention in the PHM community. A number of machine learning techniques, especially neural networks, have been applied successfully to estimate the RUL of diverse mechanical systems. ANNs have demonstrated good performance in modeling highly nonlinear, complex, multi-dimensional systems without any prior knowledge on the system behavior [14]. While the confidence limits for the RUL predictions cannot be analytically provided [15], the neural network approaches are promising for prognostic problems.

Neural networks for estimating the RUL of jet engines have been previously explored in [16] where the authors propose a multi-layer perceptron (MLP) coupled with a feature extraction (FE) method and a time window for the generation of the features for the MLP. In the publication, the authors demonstrate that a moving window combined with a suitable feature extractor can improve the RUL prediction as compared with the studies with other similar methods in the literature. In [14], the authors explore a deep learning ANN architecture, the so-called convolutional

neural networks (CNNs), where they demonstrate that by using a CNN without any pooling layers coupled with a time window, the predicted RUL is further improved.

In this paper we propose a novel framework for estimating the RUL of complex mechanical systems. The framework consists of a MLP to estimate the RUL of the system, coupled with an evolutionary algorithm for the fine tuning of data-related parameters, i.e. parameters that define the shape and quality of the features used by the MLP. The publicly available NASA CMAPS dataset [17] is used to assess the efficiency and reliability of the proposed framework. This approach allows for a simple and small MLP to obtain better results than those reported in the current literature while using less computing power.

The remainder of this paper is organized as follows. The CMAPS dataset is presented in Section 2. The framework and its components are thoroughly reviewed in Section 3. The method is evaluated using the CMAPS dataset in Section 4. A comparison with the state-of-the-art is also provided. Finally, the conclusions are presented in Section 5.

2. NASA C-MAPSS Dataset

The NASA CMAPS dataset is used to evaluate performance of the proposed method [17]. The CMAPS dataset contains simulated data produced using a model based simulation program developed by NASA. The dataset is further divided into 4 subsets composed of multi-variate temporal data obtained from 21 sensors.

For each of the 4 subsets, a training and a test set are provided. The training sets include run-to-failure sensor records of multiple aero-engines collected under different operational conditions and fault modes as described in Table 1.

The data is arranged in an $n \times 26$ matrix where n is the number of data points in each subset. The first two variables represent the engine and cycle numbers, respectively. The following three variables are operational settings which correspond to the conditions in Table 1 and have a substantial effect on the engine performance. The remaining variables represent the 21 sensor readings that contain the information about the engine degradation over time.

Each trajectory within the training and test sets represents the life cycles of the engine. Each engine is simulated with different initial health conditions, i.e. no initial faults. For each trajectory of an engine the last data entry corresponds to the cycle at which the engine is found faulty. On the other hand, the trajectories of the test sets terminate at some point prior to failure, hence the need to predict the remaining useful life. The aim of the MLP NN model is to predict the RUL of each engine in the test set. The actual RUL values of test trajectories are also included

in the dataset for verification. Further discussions of the dataset and details on how the data is generated can be found in [18].

2.1. Performance Metrics

To evaluate the performance of the proposed approach on the CMAPS dataset, we make use of two scoring indicators, namely the Root Mean Squared Error (RMSE) denoted as $e_{rms}(d)$ and a score proposed in [18] which we refer as the RUL Health Score (RHS) denoted as $s_{rh}(d)$. The two scores are defined as follows,

$$e_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} d_i^2} \tag{1}$$

$$\begin{aligned} s_{rh} &= \frac{1}{N}\sum_{i=1}^{N} s_i \\ s_i &= \begin{cases} e^{-\frac{d_i}{13}} - 1, & d_i < 0 \\ e^{\frac{d_i}{10}} - 1, & d_i \geq 0, \end{cases} \end{aligned} \tag{2}$$

where N is the total number of samples in the test set and $d = \hat{y} - y$ is the error between the estimated RUL values $\hat{y}$, and the actual RUL values y for each engine within the test set. It is important to note that $s_{rh}(d)$ penalizes late predictions more than early predictions since usually late predictions lead to more severe consequences in fields such as aerospace.

3. Framework Description

In this section, the proposed ANN-EA based method for prognostics is presented. The model consists of a multi-layer perceptron (MLP) as the main regressor for estimating the RUL of the engines in the CMAPS dataset. For the training sets, the feature vectors are generated by using a moving time window while a label vector is generated with the RUL of the engine. The label has a constant RUL for the early cycles of the simulation, and becomes a linearly decreasing function of the cycle in the remaining cycles. This is the so-called piecewise linear degradation model [19]. For the test set, a time window is taken from the last sensor readings of the engine. The data of the test set is used to predict the RUL of the engine.

The window-size n_w, window-stride n_s, and early-RUL R_e are data-related parameters, which for the sake of clarity and formalism in this study, form a vector

$v \in \mathbb{Z}^3$ such that $v = (n_w, n_s, R_e)$. The vector v has a considerable impact on the quality of the predictions by the regressor. It is computationally intensive to find the best parameters of v given the search space inherent to these parameters. In this paper, we propose an evolutionary algorithm to optimize the data-related parameters v. The optimized parameter set v allows the use of a simple neural network architecture while attaining better results in terms of the quality of the predictions compared with the results by other methods in the literature.

3.1. The Network Architecture

After careful examinations of the CMAPS dataset, we propose to use a rather simple MLP architecture for all the four subsets of the data. The implementations are done in Python using the Keras/Tensorflow environment. The source code is publicly available at the git repository `https://github.com/dlaredo/NASA_RUL_-CMAPS-`.

The choice of the network architecture is made by following an iterative process: comparing 6 different architectures, training each for 100 iterations using a mini-batch size of 512 and averaging their results over 10 different runs. Also, L1 (Lasso) and L2 regularization (Ridge) [20] are used to prevent overfitting. L1 regularization penalizes the sum of the absolute value of the weights and biases of the networks, while L2 regularization penalizes the sum of the squared value of the weights and biases. Two objectives are pursued during the iterations: 1) the architecture must be minimal in terms of layers and neurons in each layer; and 2) the performance indicators must be minimized.

The process for choosing the network architecture is as follows. First, choose a v for the experiment, say $v = (30, 1, 140)$. Next, six different ANN architectures are defined. These architectures are provided in the appendix. For each of the six different architectures, its performance is assessed using a cross-validation set from subset 1 of CMAPS. Table 2 summarizes the results for each tested architecture, while Table 3 presents the architecture chosen for the remainder of this work. The chosen architecture provides the best compromise between compactness and performance among the tested architectures.

3.2. Shaping the Data

This section covers the data preprocessing applied to the raw sensor readings in each of the datasets. Although the original datasets contain 21 different sensor readings, some of the sensors do not present much variance or convey redundant information. These sensors are therefore discarded. In the end, only 14 sensor readings out of the 21 are considered for this study. Their indices are $\{2, 3, 4, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20, 21\}$. The raw measurements are then used to create the strided time windows with window size n_w and window stride n_s. For the training labels, R_e is used at the early

stages and then the RUL is linearly decreased. The data is normalized to be within the range $[-1, 1]$ using the min-max normalization.

$$\hat{x}_i = 2 * \frac{x_i - min(x_i)}{max(x_i) - min(x_i)} - 1, \tag{3}$$

where x_i denotes the m-dimensional vector whose components are all the readings for the *i-th* sensor and $\hat{x}_i$ is the normalized x_i vector.

3.2.1. Time Window and Stride

In multivariate time-series problems such as RUL, more information can be generally obtained from the temporal sequence of the data as compared with the multivariate data point at a single time stamp. For a time window of size n_w with a stride $n_s = 1$, all the sensor readings in the time window form a feature vector $\mathbf{x}$. This approach has successfully been tested in [14, 16] where the authors propose the use of a moving window with sizes ranging from 20 to 30. In this paper, we propose not only the use of a moving time window, but also a *strided* time window that updates more than one elements ($n_s > 1$) at the time. A graphical depiction of the strided time window is shown in Figure 1.

The use of a strided time window allows for the regressor to take advantage not only of the previous information, but also to control the ratio at which the algorithm is fed with new information. With the usual time window approach, only one point is updated for every new time window. The strided time window considered in this study allows for updating more than one point at the time for the algorithm to make use of the new information with less iterations. It is believed that the information contained in the time window with stride size $n_s > 1$ is likely richer than the one contained in a time window with stride size $n_s = 1$.

3.2.2. Piecewise Linear Degradation Model

Different from common regression problems, the desired output value of the input data is difficult to determine for a RUL problem. It is usually impossible to evaluate the precise health condition and estimate the RUL of the system at each time step without an accurate physics based model. For this popular dataset, a piece-wise linear degradation model has been proposed in [19]. The model assumes that the engines have a constant RUL label in the early cycles, and then the RUL starts degrading linearly until it reaches 0 as shown in Figure 2. The piecewise linear degradation assumption is used in this work. We denote the value of the RUL in the early cycles as R_e. Initially, R_e is randomly chosen between 95 and 140 cycles. When the difference between the cycle count in the time window and the terminating cycle

of the training data is less than the initial value of R_e, R_e begins the linear descent toward the terminating cycle.

3.3. Optimal Data Parameters

As mentioned in the previous sections the choice of the data-related parameters v has a large impact on the performance of the regressor. In this section, we present the framework for picking the optimal combination of the data-related parameters n_w, n_s and R_e while being computationally efficient.

Recall that $v = (n_w, n_s, R_e)$ specific to the CMAPS dataset are bounded such that $n_w \in [1, b]$, $n_s \in [1, 10]$, and $R_e \in [90, 140]$, where all the variables are integer. The value of b is different for different subsets of the data, Table 4 shows the different values of b for each subset.

Let $X(v)$ be the training/cross-validate/test sets parametrized by v and used by the MLP to perform the RUL estimation. Finally, let $f(v) = e_{rms}(X(v))$. Recall from Equation (1) that $d = \hat{y} - y$ and that $\hat{y}$ depends on $X(v)$. Note that one function evaluation of $f(v)$ implies training the MLP and computing the result of Equation (1). Here we propose to optimize v such that

$$\min_{v \in \mathbb{Z}^3} f(v) \tag{4}$$

The problem to find optimal data-related parameters has no analytical descriptions. Therefore, no gradient information is available. An evolutionary algorithm is the natural choice for this optimization problem.

3.3.1. True Optimal Data Parameters

The finite size of CMAPS dataset and finite search space of v allow an exhaustive search to be performed in order to find the true optimal data-related parameters. We would like to emphasize that although exhaustive search is a possibility for CMAPS dataset, it is in no way a possibility in a more general setting. Nevertheless, the possibility to perform exhaustive search on the CMAPS dataset can be exploited to demonstrate the accuracy of the chosen EA and of the framework overall. In the following studies, we use the results and computational efforts of the exhaustive search as benchmarks to examine the accuracy and efficiency of the proposed approach.

We should note that the subsets of the data FD001 and FD003 have similar features and that the subsets FD002 and FD004 have similar features. Because of this, we have decided to just optimize the data-related parameters by considering the subsets FD001 and FD002 only. An exhaustive search is performed to find the true optimal values for v. The MLP is only trained for 20 epochs. Table 5 shows the optimal as well as the worst combinations of data-related parameters and the total

number of function evaluations used by the exhaustive search. It is important to notice that for this experiment the window size is limited to be larger than or equal to 15.

3.3.2. Evolutionary Algorithm for Optimal Data Parameters

Evolutionary algorithms (EAs) are a family of methods for optimization problems. The methods do not make any assumptions about the problem, treating it as a black box that merely provides a measure of quality given a candidate solution. Furthermore, EAs do not require the gradient when searching for optimal solutions, making them very suitable for applications such as neural networks.

For the current application, the differential evolution (DE) method is chosen as the optimization algorithm [21]. Though other meta-heuristic algorithms may also be suitable for this application, the DE has been stablished itself as one of the most reliable, robust and easy to use EAs. Furthermore, a ready to use Python implementation is available through the scipy package [22]. Although the DE method does not have special operators for treating integer variables, a very simple modification to the algorithm, i.e. rounding every component of a candidate solution to its nearest integer, is used for this work.

As mentioned earlier, evolutionary algorithms such as the DE use several function evaluations when searching for the optimal solutions. Recall that for this application, one function evaluation implies retraining the neural network from scratch. This is not a desirable scenario, as obtaining the optimal data-related parameters would entail an extensive computational effort. Instead of running the DE for several iterations and with a large population size, we propose to run it just for 30 iterations, i.e. the generations in the literature of evolutionary computation, with a population size of 12, which seems reasonable given the size of the search space of v.

During the optimization, the MLP is trained only 20 epochs. The small number of epochs of training the MLP is reasonable in this case because a small batch of data is used in the training, because we only look for the trend of the scoring indicators. Furthermore, it is common to observe that the parameters leading to lower score values in the early stages of the training are more likely to provide better performance after more epochs of trainings. The settings of the DE algorithm to find the optimal data-related parameters are listed in Table 6.

The optimal data-related parameters for the subsets FD001 and FD002 found by the DE algorithm are listed in Table 7. As can be observed, the results are in fact very close to the true optimal ones in Table 5 for both the subsets of the data. The computational effort is reduced by one order of magnitude when using the DE method as compared to the exhaustive search for the true optimal parameters. From

the results in Table 7, it can be observed that the maximum allowable time window is always preferred while, on the other hand, small window strides yield better results. For the case of early RUL, it can be observed that larger values of R_e are favored.

3.4. The Estimation Algorithm

Having described the major building blocks of the proposed method, we now introduce the complete framework in the form of Algorithm 1.

Algorithm 1
ANN-EA RUL Estimation Framework

Input: Initial set of data-related parameters $v \in \mathbb{Z}^n$, Raw training/testing data X and training labels y

Output: Optimal set of data-related parameters v^*

1: Choose regressor architecture (ANN, SVM, linear/logistic regression, etc).
2: Define $f(v)$ as in Section3.3.
3: Optimize $f(v)$ using the preferred evolutionary algorithm, i.e. differential evolution, evolutionary strategies, genetic algorithm, etc, using the proposed guidelines from Section 3.3.2.
4: Use v^* to train the regressor for as many epochs as needed.

4. Evaluation of the Proposed Method

In this section, we evaluate the performance of the proposed method. The architecture of the MLP is described in Table 3. The MLP was trained 10 times for 200 epochs each and tested in each subset of the CMAPS dataset. The combinations of the optimal window size n_w, window stride n_s and early RUL R_e are presented in Table 8.

The obtained results for $f(v)$ using the above setting are presented in Table 9. Notice that the performances obtained for datasets FD001 and FD002 are improved as compared with the results in Table 7. This is due to the fact that the MLP is trained for more epochs, thus obtaining better results.

The performance of the proposed method is compared against other state-of-the-art methods. Most of the methods chosen to compare here have only reported the results on the test set FD001 in terms of e_{rms}. The results are shown in Table 10. The e_{rms} value of the proposed method in Table 10 is the mean value of 10 independent runs. The values of other methods are identical to those reported in their respective original papers.

From the comparison studies of the prediction results, we can conclude that the proposed method performs better than the majority of the chosen methods when taking into consideration the whole dataset FD001. Two existing methods come close to the performance of the proposed approach in this paper, namely the time window ANN [16] and the Networks Ensemble [23]. While the performance of these two methods comes close to the results of the proposed method in this paper, the proposed method is more computationally efficient. Furthermore, the framework proposed herein is simple to understand and implement, robust, generic and light-weight. These are the features important to highlight when comparing the proposed method against other state-of-the-art approaches.

5. Conclusions

We have presented a novel framework for predicting the RUL of mechanical components. While the method has been tested on the jet-engine dataset CMAPS, the method is general enough that it can be applied to other similar systems. The framework makes use of a strided moving time window to generate the training and test records. A shallow MLP to make the predictions of the RUL has been found to be sufficient for the current dataset. The evolutionary algorithm DE needs to be run just once to find the best data-related parameters that optimize the scoring functions. The results presented in this paper demonstrate that the proposed framework is accurate and computationally efficient, which makes this framework suitable for applications that have limited computational resources such as embedded systems. Furthermore, the comparison with other state-of-the-art methods has shown that the proposed method is the best overall performer.

Two major features of the proposed framework are its generality and scalability. While for this study, specific regressors and evolutionary algorithms are chosen, many other combinations are possible and may be more suitable for different applications. Furthermore, the framework can, in principle, be used for model-construction, i.e. generating the best possible neural network architecture tailored to a specific application.

References

[1] N. Z. Gebraeel, M. A. Lawley, R. Liu, J. K. Ryan, Residual-life distributions from component degradation signals: a bayesian approach, IEEE Transactions 37 (6) (2005) 543–557.

[2] M. Zaidan, A. Mills, R. Harrison, Bayesian framework for aerospace gas turbine engine prognostics, in: IEEE (Ed.), Aerospace Conference, 2013, pp. 1–8.

[3] Z. Zhao, L. Bin, X. Wang, W. Lu, Remaining useful life prediction of aircraft engine based on degradation pattern learning, Reliability Engineering & System Safety 164 (2017) 74–83.

[4] J. Lee, F. Wu, W. Zhao, M. Ghaffari, L. Liao, D. Siegel, Prognostics and health management design for rotary machinery systems - reviews, methodology and applications, Mechanical Systems and Signal Processing 42 (12) (2014) 314–334.

[5] W. Yu, H. Kuffi, A new stress-based fatigue life model for ball bearings, Tribology Transactions 44 (1) (2001) 11–18.

[6] J. Liu, G. Wang, A multi-state predictor with a variable input pattern for system state forecasting, Mechanical Systems and Signal Processing 23 (5) (2009) 1586–1599.

[7] A. Mosallam, K. Medjaher, N. Zerhouni, Nonparametric time series modelling for industrial prognostics and health management, The International Journal of Advanced Manufacturing Technology 69 (5) (2013) 1685–1699.

[8] M. Pecht, Jaai, A prognostics and health management roadmap for information and electronics rich-systems, Microelectronics Reliability 50 (3) (2010) 317–323.

[9] J. Liu, M. Wang, Y. Yang, A data-model-fusion prognostic framework for dynamic system state forecasting, Engineering Applications of Artificial Intelligence 25 (4) (2012) 814–823.

[10] Y. Qian, R. Yan, R. X. Gao, A multi-time scale approach to remaining useful life prediction in rolling bearing, Mechanical Systems and Signal Processing 83 (2017) 549–567.

[11] N. Z. Gebraeel, M. A. Lawley, R. Li, V. Parmeshwaran, Residual-life distributions from component degradation signals: a neural-network approach, IEEE Transactions on Industrial Electronics 51 (3) (2004) 150–172.

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

[13] M. Dong, D. He, A segmental hidden semi-markov model (hsmm)-based diagnostics and prognostics framework and methodology, Mechanical Systems and Signal Processing 21 (5) (2007) 2248–2266.

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

[15] J. Z. Sikorska, M. Hodkiewicz, L. Ma, Prognostic modelling options for remaining useful life estimation by industry, Mechanical Systems and Signal Processing 25 (5) (2011) 1803–1836.

[16] P. Lim, C. K. Goh, K. C. Tan, A time window neural networks based framework for remaining useful life estimation, in: Proceedings International Joint Conference on Neural Networks, 2016, pp. 1746–1753.

[17] A. Saxena, K. Goebel, Phm08 challenge data set, [Online] Available at: https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/.

[18] A. Saxena, K. Goebel, D. Simon, N. Eklund, Damage propagation modeling for aircraft engine run-to-failure simulation, in: IEEE (Ed.), International Conference On Prognostics and Health Management, 2008, pp. 1–9.

[19] E. Ramasso, Investigating computational geometry for failure prognostics, International Journal of Prognostics and Health Management 5 (1) (2014) 1–18.

[20] P. Bühlmann, G. S., Statistics for High Dimensional Data. Methods, Theory and Applications, Springer, 2011.

[21] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359. doi:10.1023/A:1008202821328.
URL https://doi.org/10.1023/A:1008202821328

[22] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open source scientific tools for Python, [Online; accessed 06/2018] (2001–).
URL http://www.scipy.org/

[23] C. Zhang, P. Lim, A. Qin, K. Tan, Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics, IEEE Transactions on Neural Networks and Learning Systems 99 (2016) 1–13.

[24] D. P. Engelbrecht, Computational Intelligence. An Introduction, Wiley, 2007.

[25] Y. Peng, H. Wang, J. Wang, D. Liu, X. Peng, A modified echo state network based remaining useful life estimation approach, in: IEEE Conference on Prognostics and Health Management, 2012, pp. 1–7.

[26] C. Louen, S. X. Ding, C. Kandler, A new framework for remaining useful life estimation using support vector machine classifier, in: Conference on Control and Fault-Tolerant Systems, 2013, pp. 228–233.

[27] G. S. Babu, P. Zhao, X. Li, Deep convolutional neural network based regression approach for estimation of remaining useful life, in: S. I. Publishing (Ed.), 21st International Conference on Database Systems for Advanced Applications, 2016, pp. 214–228.

Table 1: C-MAPSS Dataset details.

	C-MAPSS			
Dataset	FD001	FD002	FD003	FD004
Training Trajectories	100	260	100	248
Test Trajectories	100	259	100	248
Operating Conditions	1	6	1	6
Fault Modes	1	1	2	2

Table 2: Results for different architectures for subset 1, 100 epochs.

	RMSE				RHS			
Tested Architecture	Min.	Max.	Avg.	STD	Min.	Max.	Avg.	STD
Architecture 1	15.51	17.15	16.22	0.49	4.60	7.66	5.98	0.91
Architecture 2	15.24	16.46	15.87	0.47	4.07	6.26	5.29	0.82
Architecture 3	15.77	17.27	16.15	0.45	5.11	8.25	5.93	0.94
Architecture 4	15.13	17.01	15.97	0.47	3.90	7.54	5.65	1.2
Architecture 5	16.39	17.14	16.81	0.23	5.19	6.58	5.98	0.42
Architecture 6	16.42	17.36	16.87	0.30	5.15	7.09	6.12	0.62

Table 3: Proposed Neural Network architecture

.

Layer	Shape	Activation	Additional Information
Fully connected	20	ReLU	$L1 = 0.1, L2 = 0.2$
Fully connected	20	ReLU	$L1 = 0.1, L2 = 0.2$
Fully connected	1	Linear	$L1 = 0.1, L2 = 0.2$

Table 4: Allowed values for b per subset.

	FD001	FD002	FD003	FD004
b	30	20	30	18

Table 5: Exhaustive search results for subsets FD001 and F002.

Dataset	argmin v	min $f(v)$	argmax v	max $f(v)$	Function evals.
FD001	$[24, 1, 127]$	15.11	$[25, 10, 94]$	85.19	8160
FD002	$[16, 1, 138]$	30.93	$[17, 10, 99]$	59.78	3060

Table 6: Differential evolution hyper-parameters.

Population Size	Generations	Strategy	MLP epochs
12	30	Best1Bin [24]	20

Table 7: Data-related parameters for each subset obtained with differential evolution.

Dataset	argmin v	min $f(v)$	Function evals.
FD001	$[24, 1, 129]$	15.24	372
FD002	$[17, 1, 139]$	30.95	372

Table 8: Data-related parameters for each subset as obtained by DE.

Dataset	n_w	n_s	R_e
FD001	24	1	129
FD002	17	1	139
FD003	24	1	129
FD004	17	1	139

Table 9: Scores for each dataset using the data-related parameters obtained by DE (Second architecture).

	RMSE				RHS			
Data Subset	min	max	avg	STD	min	max	avg	STD
FD001	14.24	14.57	14.39	0.11	3.25	3.58	3.37	0.11
FD002	28.90	29.23	29.09	0.11	45.99	53.90	50.69	2.17
FD003	14.74	16.18	15.42	0.50	4.36	6.85	5.33	0.95
FD004	33.25	35.10	34.74	0.53	58.52	78.62	74.77	5.88

Table 10: Performance comparison of the proposed method and the latest related papers on the CMAPS dataset.

Method	e_{rms}
ESN trained by Kalman Filter [25]	63.45
Support Vector Machine Classifier [26]	29.82
Time Window Neural Network [16]	15.16
Multi-objective deep belief networks ensemble [23]	15.04
Deep Convolutional Neural Network [27]	18.45
Proposed method with $n_w = 30$, $n_s = 1$ and $R_e = 128$	14.39

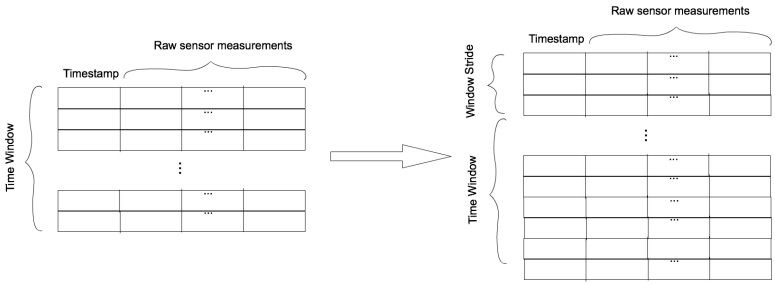

Figure 1: Graphical depiction of the time window used in this framework.

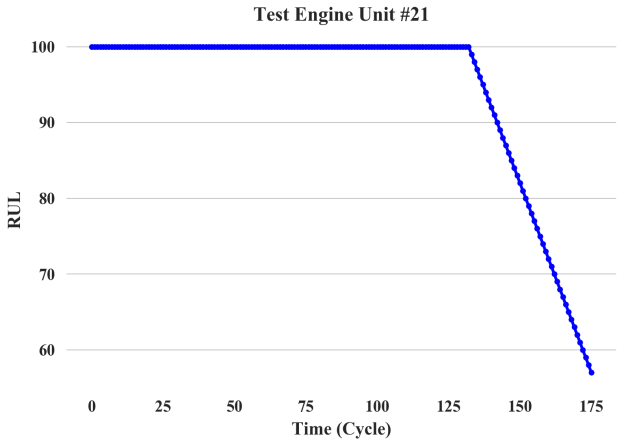

Figure 2: Piecewise linear degradation for RUL.

Tested Neural Network Architectures

Table .11: Proposed Neural Network architecture 1.

Layer	Shape	Activation	Additional Information
Fully connected	20	ReLU	L1 = 0.1, L2 = 0.2
Fully connected	20	ReLU	L1 = 0.1, L2 = 0.2
Fully connected	1	Linear	L1 = 0.1, L2 = 0.2

Table .12: Proposed Neural Network architecture 2.

Layer	Shape	Activation	Additional Information
Fully connected	50	ReLU	L1 = 0.1, L2 = 0.2
Fully connected	20	ReLU	L1 = 0.1, L2 = 0.2
Fully connected	1	Linear	L1 = 0.1, L2 = 0.2

Table .13: Proposed Neural Network architecture 3.

Layer	Shape	Activation	Additional Information
Fully connected	100	ReLU	L1 = 0.1, L2 = 0.2
Fully connected	50	ReLU	L1 = 0.1, L2 = 0.2
Fully connected	1	Linear	L1 = 0.1, L2 = 0.2

Table .14: Proposed Neural Network architecture 4.

Layer	Shape	Activation	Additional Information
Fully connected	250	ReLU	L1 = 0.1, L2 = 0.2
Fully connected	50	ReLU	L1 = 0.1, L2 = 0.2
Fully connected	1	Linear	L1 = 0.1, L2 = 0.2

Table .15: Proposed Neural Network architecture 5.

Layer	Shape	Activation	Additional Information
Fully connected	20	ReLU	L1 = 0.1, L2 = 0.2
Fully connected	1	Linear	L1 = 0.1, L2 = 0.2

Table .16: Proposed Neural Network architecture 6.

Layer	Shape	Activation	Additional Information
Fully connected	10	ReLU	L1 = 0.1, L2 = 0.2
Fully connected	1	Linear	L1 = 0.1, L2 = 0.2