A Neural Network-Evolutionary Computation framework for Remaining Useful Life Estimation

David Laredo*[1], Zhaoyin Chen[1], Oliver Schütze[2], and Jian-qiao Sun[1]

[1]School of Mechanical Engineering, University of California, Merced
[2]Department of Computer Science, CINVESTAV, Mexico City, Mexico

Abstract

This paper presents a data-driven framework for estimating the Remaining Useful Life (RUL) of mechanical systems. Two major components make up the framework: a multi-layer perceptron as base regressor and an evolutionary computation algorithm for the tuning of data-related parameters. On the data side, the framework makes use of a strided time window along with a piecewise linear model to estimate the RUL label for each time window within the training sets. Tuning the data-related parameters, i.e. window size, window stride and RUL label for the early stages of the engine, using the optimization framework here presented allows for the use of simple regressor model, i.e. Neural Networks with few hidden layers and few neurons at each layer, which can in turn allow for the use of such models in environments with very limited resources such as embedded systems. The proposed method is evaluated on the publicly available C-MAPSS dataset. The accuracy of the proposed method is compared against other state-of-the art methods available in the literature and it is shown to perform better while making use of a simpler, compact model.

Index terms— Artificial Neural Networks (ANN), Moving Time Window, RUL Estimation, C-MAPSS, Prognostics

1. Introduction

Traditionally, maintenance of mechanical systems has been carried out based on scheduling strategies, nevertheless 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 (PMH) allows for maintenance based on the current health of the system, thus cutting costs and increasing the reliability of the system [3]. To avoid confusion, here we define prognostics as the estimation of remaining useful component life. The Remaining Useful Life (RUL) of a system can be estimated based on history trajectory

*dlaredorazo@ucmerced.edu

data, this approach which we refer here as data-driven can help improve maintenance schedules to avoid engineering failures and save costs [4].

The existing PMH 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], nevertheless this comes at the expense of having extensive a prior knowledge of the underlying system and having a fine-grained model of such system (which usually involve expensive computations). On the other hand data-driven approaches tend to use pattern recognition to detect changes in system states. Data-driven approaches are appropriate when the understanding of first principles of system operation is not comprehensive or when the system is sufficiently complex (i.e. jet engines, car engines, complex machinery) such that developing an accurate model is prohibitively expensive. 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 and good prognostics results have been achieved, among the most popular algorithms we can find Artificial Neural Networks (ANN) [11], Support Vector Machine (SVM) [12], Markov Hidden Chains (MHC) [13].

Over the past few years, data-driven approaches have gained more attention in the PMH community. A number of machine learning techniques, especially neural networks have been successfully applied to the estimate RUL of diverse mechanical systems. ANNs have demonstrated good performance when applied for modeling highly nonlinear, complex, multi-dimensional system without any prior expertise on the system's physical behavior [14]. While the confidence limits for the RUL predictions can not be naturally provided [15], the neural network approaches are promising on prognostic problems.

Neural Networks for estimating the RUL of jet engines has been previously explored in [17] 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 extractior can improve the RUL prediction reported by other similar methods in the literature. In [14] the authors explore an even newer 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 Multi-layer Perceptron (MLP) to estimate the RUL of the system at hand, 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 C-MAPSS dataset [16] is used to assess the efficiency and reliability of the proposed framework. This approach allows for even simple and small MLPs, thus being suitable for computationally restricted environments, 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 C-MAPSS dataset is presented in Section 2, then the framework and all of its components are thoroughly reviewed in Section 3. The method is evaluated using the C-MAPSS dataset in Section 4, a comparison with the

state-of-the-art is also provided. Finally, our conclusions are presented in Section 5.

2. NASA C-MAPSS Dataset

The NASA C-MAPSS dataset [16] is used to evaluate performance of the proposed method. The C-MAPSS dataset contains simulated data produced using a model based simulation program (Commercial Modular Aero-Propulsion System Simulation) 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 is 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.

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

Table 1: C-MAPSS Dataset details

The data is arranged in an $n \times 26$ matrix where n corresponds to 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 operating conditions in Table 1 and have a substantial effect on engine performance. The remaining variables represent the 21 sensor readings that model the engine degradation throughout time.

Each trajectory within the train and test trajectories is assumed to represent the life-cycle of an engine. Each engine is simulated with different initial health conditions (no faults). For each trajectory of an engine the last data entry corresponds to the moment the engine is declared faulty. On the other hand the trajectories within the test sets terminate at some point prior to failure. The aim of the regressor, e.g. MLP, is then to predict the Remaining Useful Life (RUL) of each engine in the test set. The actual RUL value of each test trajectories are also included in the dataset for verification purposes. Further discussion of the dataset and details on how the data is generated are given in [18].

2.1. Performance evaluation

The evaluate the performance of the proposed approach on the C-MAPSS dataset we make use of two scoring indicators, namely the Root Mean Squared Error (RMSE) and a score function proposed in [18] which we refer in this work as RUL Health Score (RHS).

The scores are defined as follows,

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

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

where N is the total number of testing data samples and $d_i = RUL_i^p - RUL_i$ is the error between the estimated RUL value and the actual RUL value for the **i-th** testing sample. It is important to notice that the (RHS) function 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. Our method uses a Multi-Layer Perceptron (MLP) as the main regressor for estimating the RUL of the engines at each subset of the C-MAPSS dataset. For the training sets, the feature vectors are generated by using a strided time window while the labels vector is generated using a constant RUL for the early cycles of the simulation and then linearly decreasing the number of remaining cycles, this is the so called piecewise linear degradation model [19] (include image of the piecewise). For the test set, a time window is taken from the last sensor readings of the engine and used to predict the RUL of the engine.

The window size n_w, window stride n_s and early RUL R_e data-related parameters have a considerable impact on the quality of the predictions made by the regressor. Hand picking the best parameters for our application is time consuming, furthermore, a grid search approach as the ones used for hyperparameter tuning in Neural Networks is computationally expensive given the search space inherent to the aforementioned parameters. In this paper we propose the use of an evolutionary algorithm to fine tune the parameters. The optimization framework here proposed allows for the use of a simple Neural Network architecture while attaining better results in terms of the quality of the predictions made than the ones in the current literature.

3.1. The Neural Network Architecture

For this study we propose to use a rather simple MLP architecture. All the implementations were used in python using the Keras/Tensorflow environment. The structure of the Network remained consisted for all the four datasets.

The choice of the network architecture was made using an iterative process; comparing 6 different architectures, training each for 100 using a mini-batch size of 512 and averaging their

results over 10 different runs. Two objectives were pursued: that the architecture was compact, e.g. in terms of layers and neurons within each layer, and that the performance indicators were minimized. The process for choosing the network architecture was as follows: First, fix the window size n_w, the window stride n_s and the early RUL R_e, for our experiment the chosen combination is depicted in Table 2.

Window Size n_w	Window Stride n_s	Early RUL R_e
30	1	140

Table 2: Data-related parameters used for choosing the ANN architecture.

For the ANN architecture an initial setup of two layers with 250 and 50 neurons respectively was chosen as the largest ANN to use for this experiment, then for each subsequent model the number of neurons at each layer was reduced by roughly a factor of 2. For each of the 6 different architectures its performance assessed using a cross-validation set from subset 1 of C-MAPSS. Table 3 summarizes the results for each tested architectures while Table 4 presents the architecture chosen for the remainder of this work. The chosen architecture provided the best compromise between compactness and performance among the rest of the tested architectures, while at the same time being able to perform good on the other datasets. Details for all of the other tested architectures are presented in Section 6.

	RMSE				RHS			
Tested Architecture	Min.	Max.	Avg.	STD	Min.	Max.	Avg.	STD
Architecture 1	15.86	17.26	16.47	0.43	5.98	10.06	7.33	1.11
Architecture 2	15.56	17.15	16.35	0.65	6.52	20.11	4.50	4.50
Architecture 3	16.07	19.18	17.67	1.12	6.91	19.18	12.78	4.72
Architecture 4	15.32	19.99	17.63	1.48	5.93	24	13.54	6.28
Architecture 5	15.70	17.24	16.37	0.49	4.84	8.57	6.35	1.25
Architecture 6	15.58	16.92	15.95	0.39	5.44	7.65	6.38	0.68

Table 3: Results for different architectures for subset 1, 100 epochs

Layer	Shape	Activation	Additional Information
Fully connected	30	ReLU	L2 regularization factor = 0.2
Fully connected	10	ReLU	L2 regularization factor = 0.2
Fully connected	1	Linear	

Table 4: Proposed Neural Network architecture

3.2. Shaping the data

This section covers the data preprocessing applied to the raw sensor readings in each of the datasets. Even-though the original datasets contains 21 different sensor readings some of the sensor do not present much variance or convey redundant information, such 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 also normalized to be within the range $[-1, 1]$ using the min-max normalization.

$$x_{norm}^{i,j} = 2\frac{x^{i,j} - x_{min}^{j}}{x_{max}^{j} - x_{min}^{j}} - 1 \tag{3}$$

where $x_{i,j}$ denotes the original i-th data point of the j-th sensor and x^{i,j_norm} is the normalized value of $x^{i,j}$. x_{max}^{j} and x_{min}^{j} denote the maximum and minimum values of the original measurement data from the j-th sensor, respectively.

3.2.1 Time Window Processing

In multivariate time-series based problems such as RUL, more information can be generally obtained from the temporal sequence of data as compared with the multivariate data point at a single time stamp. Let n_w denote the size of the time window, for a time window with a stride $n_s = 1$, all the past sensor values within the time window are collected and put together to form a feature vector $\mathbf{x}$. This approach has successfully been tested in [14, 17] where they propose the use of a moving window with values raging 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 n_s elements at the time instead of 1. A graphical depiction of the strided time window is shown in Figure 1.

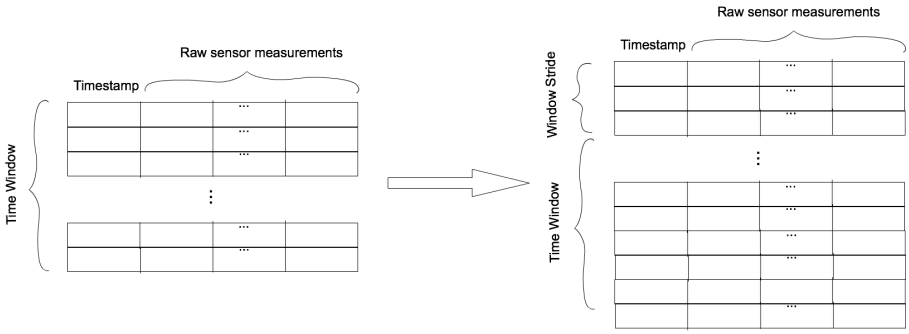

Figure 1: Graphical depiction of the time window used in this framework.

The use of a *strided time window* allows for the regressor to take advantage not only of the previous information available, 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, on the contrary, the strided time window allows for updating n_s points at the time, allowing for the algorithm to catch newer information with fewer iterations, furthermore, the information contained in the strided time window is likely more rich than the one contained in a time window with stride of 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 piece-wise linear degradation 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 approach is used for this work, in here we denote the value for the RUL at the early stages as R_e.

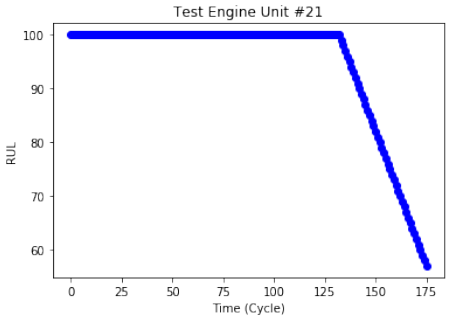

Figure 2: Piecewise linear degradation for RUL.

3.3. Choosing optimal data-related parameters

As mentioned in the previous sections the choice of the data-related parameters window size n_w, window stride n_s and constant RUL R_e have a large impact on the performance of the regressor, i.e. the MLP. In this section we present a framework for picking the best combination of the data-related parameters n_w, n_s and R_e without spending too much computational time.

Let $\mathbf{v} = (n_w, n_s, R_e)$, where $n_w \in [1, b]$, $n_s \in [1, 10]$ and $R_e \in [90, 140]$ and all the intervals are integer intervals. The value of b is dependent upon the specific subset, Table 5 presents the different values b can take for each dataset.

	FD001	FD002 2	FD003	FD004
b	30	20	30	18

Table 5: Allowed values for b per subset

Given $\mathbf{v}$, we can evaluate the performance of the regressor by reshaping the data using $\mathbf{v}$, training the MLP using the obtained data and then computing the scores in equations 1 and 2.This setting is just what is required for performing any kind of optimization, i.e. to have a set of tunable parameters and a performance indicator, therefore, here we propose to fine tune $\mathbf{v}$ using a meta-heuristic algorithm.

3.3.1 Differential Evolution for obtaining the optimal data-related parameters

Differential Evolution (DE) [20] is a method that optmizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. The method does not make any assumptions about the problem, therefore it is know as a metaheuristic, nevertheless, this kind of methods are not guaranteed to converge to the optimal solution. DE does not require the gradient of the problem being optimized, making it a very suitable metaheuristic for applications such as Neural Networks.

DE belongs to a class of algorithms known as evolutionary algorithms. The algorithm optimizes a problem by maintaining a *population* of candidate solutions and creating new candidate solutions $\mathbf{v}'$ by combining existing ones according to a simple cross-over formula, keeping whichever candidate solution $\mathbf{v}^*$ has the best score or fitness on the optimization problem at hand. In this way the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed.

Although DE does not have special operators for treating integer variables a very simple modification to the algorithm, consisting on rounding every candidate solution $\mathbf{v}'$ to the nearest integer, is used for this work.

There is yet one last detail to be taken care of, evolutionary algorithms such as DE tend to use several function evaluations for obtaining the optimal solutions, for our application one function evaluations implies retraining the Neural Network. Certainly this is not a desirable scenario, as obtaining the optimal parameter vector $\mathbf{v}$ would entail an extensive use of computational power. Instead of running for DE for several iterations and with a large population we propose to run it just for 10 iterations (generations in the literature of evolutionary computation) and using a population size of 10, which seems reasonable given the size of the search space of $\mathbf{v}$. Furthermore, during the tuning process the MLP is not trained for 100 epochs, instead the MLP is trained for just 20 epochs, this is done mainly for two reasons: the use of the mini-batch in the training process allows for a speed up in the convergence, therefore it can be assumed that the algorithm will most likely be very close to its optima after just a couple of iterations, second and most important is the fact that we assume that parameters that lead to lower score values in the early stages of the MLP training process are more likely to provide better performance overall. Given the similarities between subsets FD001/FD003 and FD002/FD004 we have decided to just tune the for subsets FD001 and FD002. Details for the use of DE in finding the optimum data-related parameters are described in Table 6. The optimal data-related parameters for each of the subsets found by DE are shown in Table 7.

Population Size	Generations	Strategy	MLP epochs
30	20	Best1Bin	20

Table 6: Differential Evolution hyper-parameters.

Dataset	Window Size n_w	Window Stride n_s	Early RUL R_e
FD001	30	2	120
FD002	20	2	120

Table 7: Data-related parameters for each subset obtained with Differential Evolution.

As can be observed in Table 7, the maximum window size n_w is preferred for each dataset, furthermore, a window stride $n_s = 2$ is preferred for both sets, finally the largets available R_e was also optimal as per the results obtained with DE.

4. Evaluating the performance of the proposed method

In this section we evaluate the performance of the proposed method. The architecture of the MLP to be used here was presented in Section 3 Table 4 and will be used throughout this section. The MLP was trained 10 times for200 epochs each and tested in each subset of the C-MAPSS dataset.

For the first experiment the combinations of window size n_w, window stride n_s and early RUL R_e presented in Table 8 were tested obtaining the results shown in Table 9.

Dataset	Window Size n_w	Window Stride n_s	Early RUL R_e
FD001	30	2	120
FD002	20	2	120
FD003	30	2	120
FD004	18	2	120

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

	RMSE				RHS			
Data Subset	Min.	Max.	Avg.	STD	Min.	Max.	Avg.	STD
FD001	14.78	15.25	14.98	0.13	3.41	4.40	3.94	0.30
FD002	29.76	31.55	30.67	0.50	59.25	95.36	69.25	10.68
FD003	15.05	16.05	15.54	0.33	3.24	4.98	3.86	0.57
FD004	34.61	37.75	35.58	0.99	55.46	91.94	69.06	11.12

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

Next, the possibility of using a single set of data-related parameters for all the subsets is explored. For this experiment the window size n_w is fixed for all of the four datasets, given that the maximum allowable window size for all datasets is 18, $n_w = 18$ will be used as window size throughout the four subsets. Details of the data-related chosen parameters are presented in Table 10, the results obtained are shown in Table 11.

Dataset	Window Size n_w	Window Stride n_s	Early RUL R_e
All	18	2	120

Table 10: Single set of data-related parameters for all subsets.

As can be observed, performance is decreased for subsets FD001/FD003, this indicates that larger window sizes are beneficial for this regression problem. Figures 3 and 4 show a comparison of how the scores are affected for each dataset by changing the data-related parameters to make use of 2 and 1 sets of them.

	RMSE				RHS			
Data Subset	Min.	Max.	Avg.	STD	Min.	Max.	Avg.	STD
FD001	17.45	19.29	18.12	0.63	6.60	8.96	7.45	0.81
FD002	29.64	31.93	30.28	0.67	49.02	95.62	60.65	14.18
FD003	15.90	16.80	16.19	0.32	3.63	5.11	4.04	0.45
FD004	33.63	36.43	34.63	0.86	52.14	78.47	61.87	8.47

Table 11: Scores for each dataset using the single set of data-related parameters.

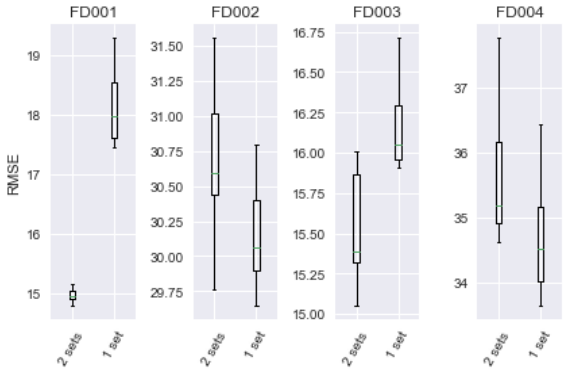

Figure 3: Comparison of RMSE results for different sets of data-related parameters.

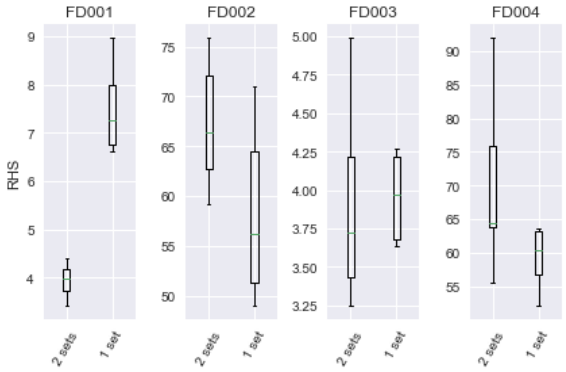

Figure 4: Comparison of RHS results for different sets of data-related parameters.

4.1. Comparison with other approaches

In this section the performance of the proposed method is compared against other state-of-the-art methods. Most of the presented methods in this section have only reported results on the test set FD001 in terms of RMSE, the results are displayed in Table 12. The RMSE value of the proposed method in Table 12 is the mean value of 10 independent runs. The remainder of the values are identical to those reported in their respective original papers.

Based on the results, the proposed method performance is better than the majority of the compared methods when taking into consideration the whole dataset FD001. The only method that outperforms the presented approach is the modified ESN [21], nevertheless, the method misses to

Method	No. of Engines	RMSE
ESN trained by Kalman Filter [21]	100 of 100	63.45
Modified approach with classified sub- models of the ESN [21]	80 of 100	7.02
Support Vector Machine Classifier [22]	100 of 100	29.82
Time Window Neural Network [17]	100 of 100	15.16
Multi-objective deep belief networks ensemble [23]	100 of 100	15.04
Deep Convolutional Neural Network [24]	100 of 100	18.45
Proposed method with $n_w = 30$**,** $n_s = 2$ **and** $R_e = 120$	100 of 100	14.90

Table 12: Performance comparisons of the proposed method and the latest related papers on the C-MAPSS dataset.

estimate the RUL for the remaining 20 engines. Two methods come close to the perfomance of the presented approach in this paper, namely the time window ANN [17] and the Networks Ensemble [23]. While the performance of both methods comes close to the results presented in this paper, the presented approach is computationally more efficient. Furthermore, the approach proposed in this paper is extremely simple to understand and implement, robust, generic, light-weight and fast to train, features that we believe are important to highlight when comparing the proposed method against other state-of-the-art approaches.

5. Conclusions

This paper presents a systematical framework for predicting the RUL of mechanical components. While the method was tested on the jet-engine specific dataset C-MAPSS the method is general enough so that it can theoretically be applied to other kind of similar systems. The framework makes use of a strided moving time window to generate the training and test sets, a shallow MLP to make the predictions of the RUL and an evolutionary algorithm (DE) which needs to be run just once in order to find the best data-related parameters that optimize the scoring functions used in this study. This study also shows that the proposed method is the best overall performer when compared to other methods in the recent literature.

6. Appendix A - Tested Neural Network Architectures

In this section we present the six tested neural network architectures

Architecture 1

Layer	Shape	Activation	Additional Information
Fully connected	30	ReLU	L2 regularization factor = 0.2
Fully connected	10	ReLU	L2 regularization factor = 0.2
Fully connected	1	Linear	

Table 13: Proposed Neural Network architecture 1

Architecture 2

Layer	Shape	Activation	Additional Information
Fully connected	50	ReLU	L2 regularization factor = 0.2
Fully connected	20	ReLU	L2 regularization factor = 0.2
Fully connected	1	Linear	

Table 14: Proposed Neural Network architecture 2

Architecture 3

Layer	Shape	Activation	Additional Information
Fully connected	100	ReLU	L2 regularization factor = 0.2
Fully connected	50	ReLU	L2 regularization factor = 0.2
Fully connected	1	Linear	

Table 15: Proposed Neural Network architecture 3

Architecture 4

Layer	Shape	Activation	Additional Information
Fully connected	250	ReLU	L2 regularization factor = 0.2
Fully connected	50	ReLU	L2 regularization factor = 0.2
Fully connected	1	Linear	

Table 16: Proposed Neural Network architecture 4

Architecture 5

Layer	Shape	Activation	Additional Information
Fully connected	20	ReLU	L2 regularization factor = 0.2
Fully connected	1	Linear	

Table 17: Proposed Neural Network architecture 5

Architecture 6

Layer	Shape	Activation	Additional Information
Fully connected	10	ReLU	L2 regularization factor = 0.2
Fully connected	1	Linear	

Table 18: Proposed Neural Network architecture 6

References

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

[2] M.A. Zaidan, A.R. Mills, and R.F. Harrison. Bayesian framework for aerospace gas turbine engine prognostics. In IEEE, editor, *Aerospace Conference*, pages 1–8, 2013.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[17] P. Lim, C. K. Goh, and K. C. Tan. A time window neural networks based framework for remaining useful life estimation. In *Proceedings International Joint Conference on Neural Networks*, pages 1746–1753, 2016.

[18] A. Saxena, K. Goebel, D. Simon, and N. Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. In IEEE, editor, *International Conference On Prognostics and Health Management*, pages 1–9, 2008.

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

[20] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997.

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

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

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

[24] G. S. Babu, P. Zhao, and X. Li. Deep convolutional neural network based regression approach for estimation of remaining useful life. In Springer International Publishing, editor, *21st International Conference on Database Systems for Advanced Applications*, pages 214–228, 2016.