Technical Report on the solution of the CMAPSS-RUL dataset using Neural Networks

David Laredo*[1] and Jian-qiao Sun†[1]

[1]School of Mechanical Engineering, University of California, Merced

Abstract

In this report we present an a data-driven approach for estimating the Remaining Useful Life (RUL) of aero-engines. A "strided" time window is employed to generate training and test sets to be used with a conventional Multi-layer Percepton (MLP) which will serve as the main regressor for this application, no model for the engine is required . 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.

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 strategies such as breakdown corrective maintenance and scheduled preventive maintenance 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 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].

*dlaredorazo@ucmerced.edu
†jqsun3@ucmerced.edu

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.

In this paper we propose a Multi-layer Perceptron (MLP) architecture coupled with a strided time-window approach for estimating the RUL of aero-engines. The publicly available NASA C-MAPSS dataset [16]. Raw sensor measurements with normalization are directly used as inputs to the MLP which then outputs the RUL of the jet engine in terms of cycles.

The use of 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 moving time-window and suitable feature extraction they 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 as described in [17] cite the predicted RUL is further improved.

The present work takes inspiration from [17] and [14] in the sense that an ANN architecture coupled with a time-window is used to produce the RUL predictions, nevertheless this research also considers the use of a *strided* time-window which allows for more accurate predictions of the RUL of the jet-engine than the reported in [17] and [14]. Furthermore, this paper presents an optimization framework, based on the Root Mean Squared Error RMSE of the predictions, for the fine tuning of data related hyperparameters such as window-size, stride-size, etc. Such approach allows a simple MLP to obtain even better results than those reported in the current literature using less computing power.

The remainder of this paper is organized as follows:...

2. NASA C-MAPSS Dataset

The NASA C-MAPSS dataset [16] is used to evaluate 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

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.

	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

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 and the aim is to predict the Remaining Useful Life (RUL) of each engine in the test set. The actual RUL value of each test trajectories were 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).

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

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

where s denotes the score and N is the total number of testing data samples. $d_i = RUL_i^p - RUL_i$, that is the error between the estimated RUL value and the actual RUL value for the **i-th** testing sample. 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. Estimating Remaining Useful Life using Multi-Layer Perceptron as Regressor

In this section the proposed ANN-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]. 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 in 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, i.e. Differential Evolution (DE) [20], 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 4 different architectures, running each 10 times for 100 epochs and using a mini-batch size of 512. Two objectives were pursued; that the size of the architecture was small enough, e.g. in terms of layers and neurons within each layer and that the performance indicators were better than the ones presented in the literature so far. The process for choosing the network architecture can be described as follows: First, fix the window size n_w, the window stride n_s and the early RUL R_e.

An initial setup of two layers with 250 and 50 neurons each was chosen, the number of neurons at each layer was then reduced by roughly a factor of 2 and tested using a cross-validation set from subset 1 of C-MAPSS. When the performance of the network started to degrade the process was stopped. Table 2 summarizes the results for each tested architecture, Table 3 presents the architecture chosen for the remainder of this work yielded the best results and hence was the chosen for the rest of the experiments. Details for the other 3 tested architectures are presented in Section ...

	Min.		Max.		Avg.		STD	
Tested Architecture	RMSE	RHS	RMSE	RHS	RMSE	RHS	RMSE	RHS
Architecture 1	10.85	151.65	12.23	277.67	11.66	226.94	0.45	41.58
Architecture 2	11.12	186.22	13.98	365.92	12.68	280.41	1.03	64.07
Architecture 3	11.58	179.15	12.72	266.55	12.04	215.09	0.35	28.39
Architecture 4	12.52	262.77	14.25	368.35	13.58	325.41	0.53	34.13

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

Layer	Shape	Activation	Additional Information
Fully connected	30	ReLU	Dropout(0.6)
Fully connected	10	ReLU	Dropout(0.2)
Fully connected	1	Linear	

Table 3: 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 and are therefore discarded. Therefore 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 labels R_e is used at the early stages and then the RUL is linearly decreased. Finally, the data is 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.

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. 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.

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 4 presents the different values b can take for each dataset.

	FD001	FD002 2	FD003	FD004
b	30	20	30	18

Table 4: 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. Details for the use of DE in finding the optimum data-related parameters are described in Table 5. The optimal data-related parameters for each of the subsets found by DE are shown in Table 6.

Population Size	Generations	Strategy	MLP epochs
10	10	Best1Bin	20

Table 5: Differential Evolution hyper-parameters.

Dataset	Window Size n_w	Window Stride n_s	Early RUL R_e
FD001	26	2	100
FD002	16	2	91
FD003	30	2	97
FD004	16	2	92

Table 6: Optimal data-related parameters for each subset.

4. Appendix

Different architectures tested

Architecture 1

Layer	Shape	Activation	Additional Information
Fully connected	30	ReLU	Dropout(0.6)
Fully connected	10	ReLU	Dropout(0.2)
Fully connected	1	Linear	

Table 7: Proposed Neural Network architecture 1

Architecture 2

Layer	Shape	Activation	Additional Information
Fully connected	50	ReLU	Dropout(0.6)
Fully connected	20	ReLU	Dropout(0.2)
Fully connected	1	Linear	

Table 8: Proposed Neural Network architecture 2

Architecture 3

Layer	Shape	Activation	Additional Information
Fully connected	100	ReLU	Dropout(0.6)
Fully connected	50	ReLU	Dropout(0.2)
Fully connected	1	Linear	

Table 9: Proposed Neural Network architecture 3

Architecture 4

Layer	Shape	Activation	Additional Information
Fully connected	250	ReLU	Dropout(0.6)
Fully connected	50	ReLU	Dropout(0.2)
Fully connected	1	Linear	

Table 10: Proposed Neural Network architecture 4

	Min.		Max.		Avg.		STD	
Tested Architecture	RMSE	RHS	RMSE	RHS	RMSE	RHS	RMSE	RHS
Architecture 1	11.09	172.18	14.98	397.79	12.96	295.51	1.27	68.62
Architecture 2	11.27	187.88	14.28	344.96	12.84	274.69	0.82	42.78
Architecture 3	12.45	257.33	15.60	483.22	14.46	389.27	1.03	78.12
Architecture 4	13.26	307.14	15.58	465.33	14.57	405.08	0.74	52.03

Table 11: Results for different architectures for subset 1, 250 epochs

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.