Robustness Testing Framework for RUL Prediction Deep LSTM Networks

Mohamed Sayah[a,*], Djillali Guebli[a], Zeina Al Masry[b], Noureddine Zerhouni[b]

[a] *University Oran1 FSEA Faculty Oran Algeria*
[b] *EMTO-ST Institute, Univ. Bourgogne Franche-Comté, CNRS, ENSMM 24 rue Alain Savary, Besançon Cedex, 25000, France*

Abstract

Efficiency and robustness in remaining useful life (RUL) prediction are crucial in system health monitoring. Thus, the internal logic computation of a Deep LSTM model for RUL prediction is mainly shaped and evaluated over a training data-set and its performance examined on a testing data-set. This paper proposes a framework for testing robustness of deep Long Short Term Memory (LSTM) architecture for remaining useful life prediction that enables to gain confidence in the trained LSTM model for RUL prediction and ensures better quality. The resiliency of proposed Deep LSTM networks for RUL estimation using stress functions is first checked then the effect of the stress on model performance is analyzed. A comparison between the performance of the constructed mutant fuzzed Deep LSTM networks and the original Deep LSTM model for RUL prediction is provided to determine the quality of the RUL prediction model.

Furthermore, the main purpose of this paper is to determine to what extent Deep LSTM models in the neighborhood of the trained LSTM model still have high test accuracy and quality scoring. Thus, the use of φ-stress operators shows that we could build stable and data-independent Deep LSTM models for RUL prediction. Finally, the proposed framework is validated using the Commercial Modular Aero Propulsion System Simulation (C-MAPSS) data-set.

Keywords: RUL prediction, LSTM model, Mutant Model, Fuzzy Deep LSTM network, Robustness, ϵ-Robustness.

*Corresponding author, sayahmh@gmail.com

1. Introduction

Predictive maintenance (PdM) is applied in complex industry areas such as aerospace systems, nuclear power plants, chemical plants, advanced military systems, automotive manufacturing, and transportation [1]. The purpose of PdM is to ensure the reliability of complex systems by reducing downtimes and preventing unplanned failures which lead to cost-saving especially when scheduled maintenance is considered unnecessary and is canceled.

This crucial phase in the Prognostic and Health Management (PHM) process for complex systems [1] is defined as condition-based maintenance, and it is performed to avoid catastrophic breakdown events. These breakdowns express operating limits of a system components, known in each domain expertise by analyzing the specific journal of failures or degradation modes [2]. The predictive maintenance process intends to monitor industrial system conditions to precisely schedule maintenance actions [3, 4]. This key objective in the health management of complex industrial systems [5] enables to determine precisely the time for a system component to perform its functional capabilities before it fails [6, 7].

For health monitoring information [2], remaining useful life (RUL) prediction of a system component is of primary interest where several RUL estimation methods were proposed in the literature [8, 9]. Thus, suitable and feasible deep data-driven models were proposed in [10, 11, 5] for health management and component prognostic in many industrial fields. These data-driven models use data-set traces of failures and operating conditions to determine precise RUL estimates for the industrial complex system components. These given ongoing data sets repositories, which are time series from some nominal states to a failed state, can not guarantee efficiency and robustness of proposed RUL prediction Deep Long Short Term Memory models [10]. The question is to what extent a Deep LSTM model is robust and how its robustness could impact RUL estimation quality for a system component. In that respect, we notice that it is well known that achieving better accuracy for a Deep LSTM model using RMSE and scoring metrics did not ensure genericity and it is not good enough for a better robustness quality evaluation [12]. Thus, a robustness testing framework should necessarily reinforce quality evaluation and confidence for a trained Deep LSTM Model.

In this paper, we propose a framework for Deep LSTM robustness testing based on prognostics data repository C-MAPSS, a collection of data sets used exclusively for prognostic of system components. After training and constructing the Deep LSTM network for RUL estimation, a mutation process is handled by invoking the stress on the internal logic of this generated deep learning network for RUL prediction. For each mutant model, a deep network is constructed and then evaluated. Finally, a smooth comparison is done between mutant models and the original learned Deep LSTM model, to qualify robustness of the RUL prediction baseline model. The remainder of the paper is structured as follows. Section 2 presents related work. Section 3 introduces the robustness testing framework and then Section 4 details Deep LSTM Fuzzy Models. In section 5, we present the robustness of the deep last model. Finally, experiments and conclusions are presented, respectively, in Sections 6 and 7.

2. Related Works

In [13], RUL estimation is determined with models built to capture time sequence features. These collected informations are time-dependent events that could be modeled as a time series in nature. For sliding models introduced in [14], the entire time sequence series of sensors data are sliced into windows characterized by their local RUL estimation. Nevertheless, the issue in such a sliding model is to better determine window sizes and so, to keep dependency over time of collected sensors data. Larger is the slide window time range more the time sequence information is captured. However, when the time range is larger, the entire sequence information is difficult to handle and over-fitting often appears. The challenge still is the preservation of time dependency between sliding windows.

Hidden Markov Models (HMMs) were proposed in [15] to deal with time dependencies in information sequences. Unfortunately, the HMM model states depend only on the last previous information which makes them inappropriate for long range dependencies [16]. Consequently, major drawbacks of hidden Markov models are related to space and time consuming, as well as their inefficiency for large size discrete data. In [8, 17], neural network techniques were implemented for short or long range time dependencies for RUL prediction. These models known as Neural Recurrent Networks (RNN) [18, 19] can efficiently handle short-term time sequence data but are limited when applied for long-term time dependency solving problems [20] and often propagated gradient either vanish or explode.

Currently, more efficient deep learning approaches have been widely used in different information technology applications such as computer vision, image and video saliency processing, speech recognition, and natural language processing [21, 22, 23]. Even more, convolutional Neural Network and Long Short-Term Memory network [22, 24, 8] were used to deal with long-term time dependency in RUL prediction using In-gate, Forget-gate, and Output-gate functions for LSTM cells. Indeed, the recent research in RUL prediction tries to leverage Deep LSTM learning, but some concerns raised about the need for more generalization and robustness of such models [12]. This is due to the inadequate configuration of the Deep LSTM network for RUL prediction but also the lack of training/testing data sets. In such proposed LSTM neural networks for RUL prediction, the key issue is robustness testing where mutation techniques can be used to stress the internal logic of the Deep LSTM model for RUL estimation and then assess its quality.

The mutation process would impact the source-level, program-level, or model-level of a deep learning network for RUL prediction. Thus, by using different mutation operators, stresses are injected in training data, training programs or even internal logic of the pre-trained deep network. Furthermore, the design of mutation operators as the key components to ensure the genericity and robustness of the trained Deep LSTM models could be used at the input source level as stress operators to enrich training/testing data sets. In the programming level, mutation-based code testing enables to maintain Deep LSTM model performance. Unfortunately, the mutation operators at source and program levels require more space and time-consuming computation. However, the model level mutation operators seem to be more suitable for our robustness framework. The core contribution is to ensure better quality evaluation of a Deep LSTM model for RUL prediction, knowing that the

trained internal layers and cells in LSTM models using training/testing data sets still have a lack of genericity and efficiency. This is because of Deep LSTM Network sensitivity to any change in training/testing data sets or program tuning parameters. In our model-based mutation robustness testing, quality of testing data sets generated by the NASA institution - *Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)* - has been of great interest in this proposed robustness framework.

3. Robustness Testing Framework

Although suitable training and testing data sets are used, a deep learning model that has achieved high accuracy results may still have a lack of generalization. Thus, the robustness of the Deep LSTM model is of great importance to gain confidence in RUL prediction data-driven models. Besides, relevant verification on a validated testing data-set is necessary for Deep LSTM model efficiency. Consequently, a well-established mutation framework to verify and evaluate the quality of a Deep LSTM model is crucial to trust deep models for RUL estimation.

In our context, we have defined a set of model-level mutation operators [25] that introduces exclusively stresses to the weights of the deep learning models without a training process. This means that training data sets and programs were kept unchanged. The aim is to determine and perform to what extent a Deep LSTM model is sensitive to model-level stresses. We notice that use of model-level mutation for RUL prediction models was motivated first, in regards to time-consuming in the training phase and second, according to deep model architecture role in solving RUL estimation problem (see Figure 1).

The proposed framework in Figure 1 shows a workflow of fuzzing or stressing transformation steps on an original Deep LSTM model for RUL prediction. The mutants are considered holistically as outputs of model-level mutation techniques that impact LSTM cell weights and different layers of RUL prediction deep model. The cell and layer weights are stressed by the appropriate φ-functions. The stress of these latters was validated first on Weierstrass function [26] and corresponding ρ stress intensities were tuned adequately (see Equation 1).

$$\begin{array}{l} W_{(a,b)}(x) = \sum_{n=0}^{k} a^n cos(b^n \pi x). \quad k \in \mathbb{N}, k > 100 \\ 0 < a < 1 \\ \text{ab} > 1 + (3/2)\pi \end{array} \tag{1}$$

One should note that Weierstrass function was adopted for stress tuning accordingly to its property of being continuous everywhere but differentiable nowhere (i.e. high stress sensitivity). Once more, *Weierstrass* $W_{(a,b)}$ function (see Figure 2) has a fractal behavior where any zoomed region has the same values of the whole function. This helped us to construct smooth and appropriate stress functions to impact gradually the Deep LSTM model for RUL prediction.

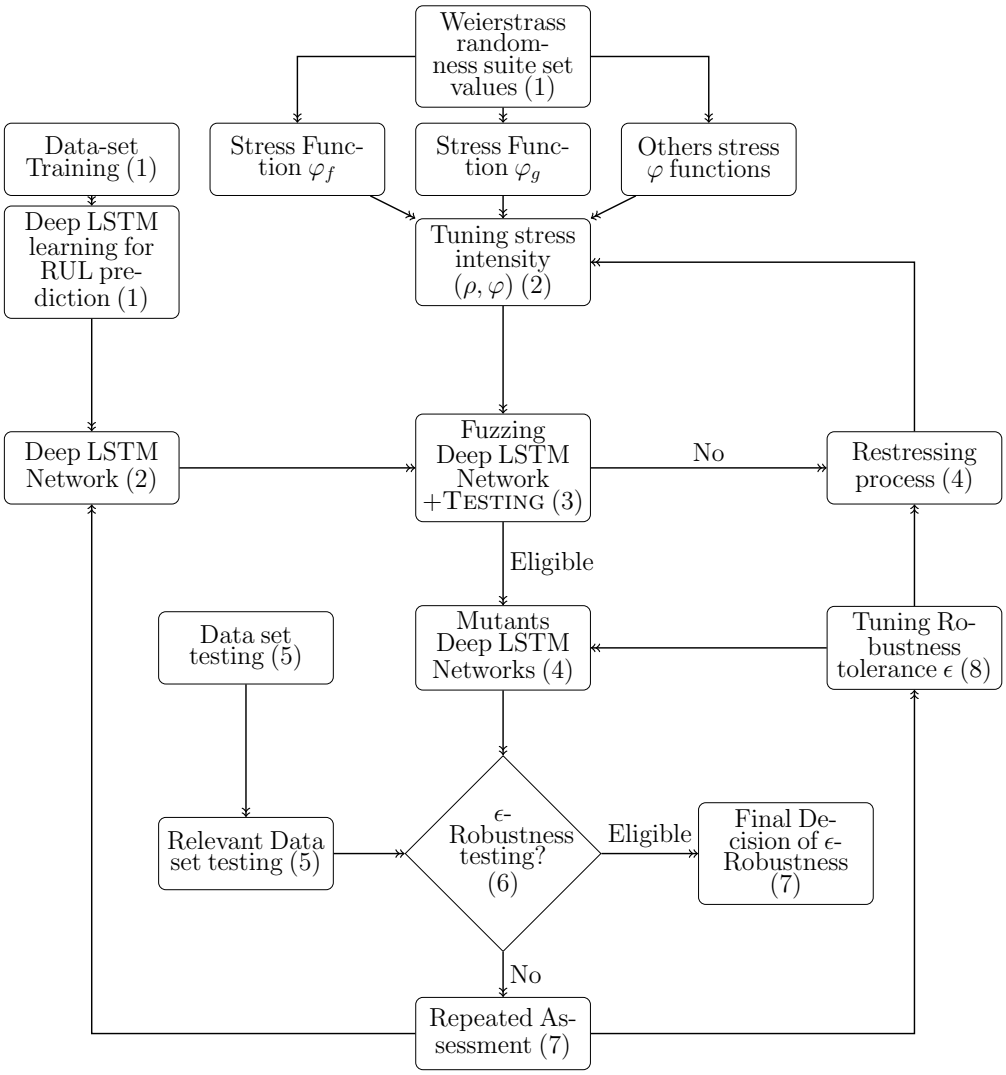

Figure 1: Robustness testing framework for a hyperparameter ϵ.

4. Fuzzy Mutants of Deep LSTM Model

Let us now consider an original pre-trained Deep LSTM model for RUL prediction with its internal logic including hidden layers L and based on LSTM cell structure. Each LSTM cell is composed of weights w_A, w'_A, w_I, w'_I, w_O, w'_O, w_F and w'_F as presented in Figure 3 and Figure 4.

As discussed previously, model-level based mutants for RUL prediction Deep LSTM model were constructed by applying φ operator stressing. These operators are validated and tuned upstream using the Weierstrass function (see Equation 1). Therefore, various model-level based mutants can be generated for an original Deep LSTM architecture [27, 28] (see Figure 3). In our case, the configurations $C_{\#}$ would be stressing of one-exclusive, two-mutually exclusive, three-mutually exclusive, or four-mutually exclusive weights (see

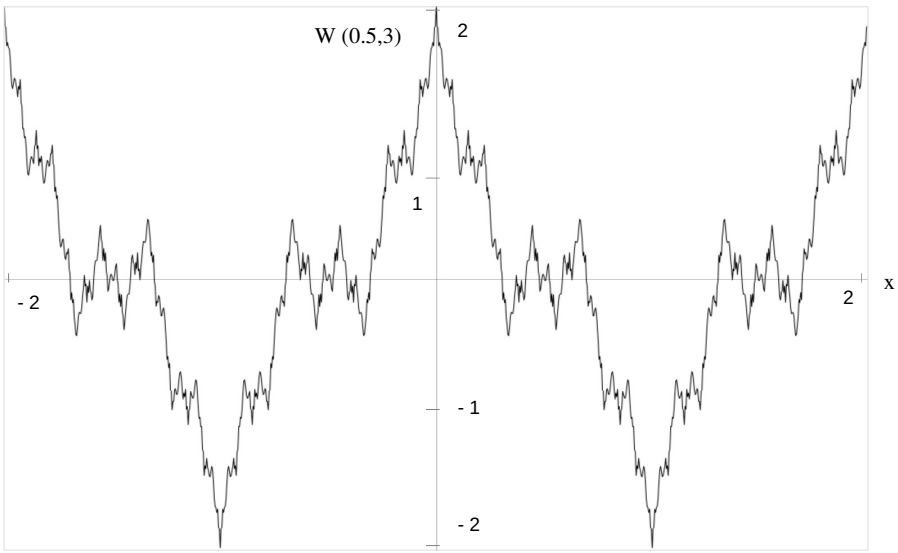

Figure 2: Weierstrass $W_{(0.5,3)}$ on interval [–2, 2].

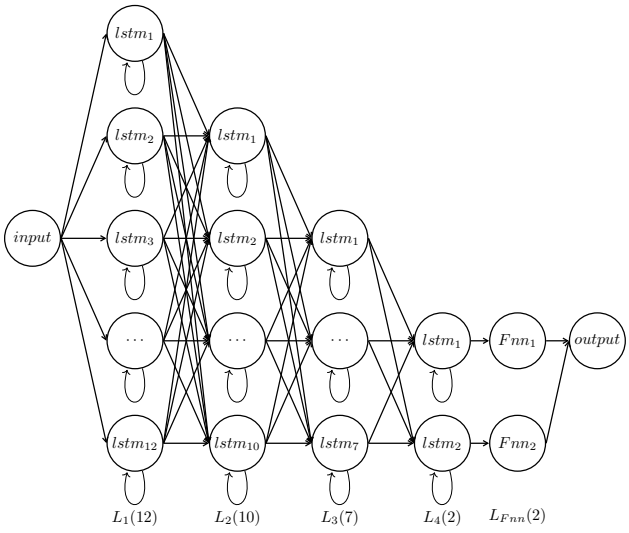

Figure 3: Deep LSTM-NN Network

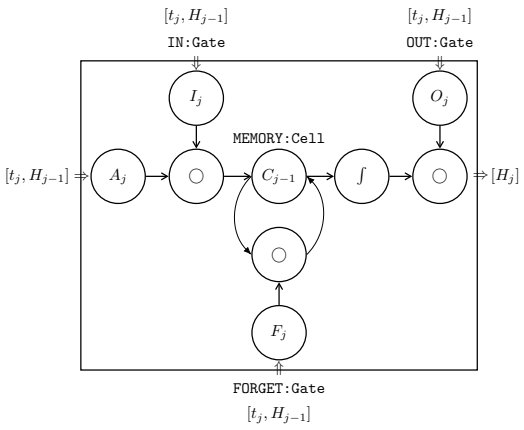

Figure 4: Standard LSTM cell

Tables 1, 2). We noticed that it is not beneficial to stress at the same time the input and the output weights of an LSTM cell component node A, I, O, F (see Figure 4). This is since the impact of stressed weights will not be consistent. Also, choco solver [29] was used to determine exactly the number of model-level based mutants of a given Deep LSTM model. Indeed, the variant configurations were respectively 8, 24, 32, and 16 mutants for one-exclusive, two-mutually exclusive, three-mutually exclusive, or four-mutually exclusive stressing.

Config. $C_{\#}$	w_A	w'_A	w_I	w'_I	w_O	w'_O	w_F	w'_F
C_1	**1**	0	0	0	0	0	0	0
C_2	0	**1**	0	0	0	0	0	0
C_3	0	0	**1**	0	0	0	0	0
C_4	0	0	0	**1**	0	0	0	0
C_5	0	0	0	0	**1**	0	0	0
C_6	0	0	0	0	0	**1**	0	0
C_7	0	0	0	0	0	0	**1**	0
C_8	0	0	0	0	0	0	0	**1**

Table 1: Model configuration for one-exclusive stressing

Config. $C_{\#}$	w_A	w'_A	w_I	w'_I	w_O	w'_O	w_F	w'_F
C_1	**1**	0	0	**1**	0	0	0	0
C_2	0	**1**	0	0	0	0	0	**1**
C_3	0	0	0	**1**	0	0	0	**1**
C_4	0	0	**1**	0	0	0	**1**	0
...	...	...	...	...	...	...	...	...
C_{22}	0	0	**1**	0	**1**	0	0	0
C_{23}	**1**	0	0	0	0	**1**	0	0
C_{24}	0	0	0	0	0	**1**	0	**1**

Table 2: Model configuration for two-exclusive stressing

In our experiments for Deep LSTM model robustness testing, we use one-exclusive stressed weights of the internal LSTM cell weights w_A, w'_A, w_I, w'_I, w_O, w'_O, w_F, and w'_F (see Equation 2). Thus, eight (08) mutant models under many stress intensities are considered (see Table 1).

$$\text{Algebric Model for LSTM cell} \left| \begin{array}{l} I_j = \sigma(t_j \times w_I + H_{j-1} \times w'_I + b_I) \\ O_j = \sigma(t_j \times w_O + H_{j-1} \times w'_O + b_O) \\ F_j = \sigma(t_j \times w_F + H_{j-1} \times w'_F + b_F) \\ A_j = \tanh(t_j \times w_A + H_{j-1} \times w'_A + b_A) \\ C_j = F_j \circ C_{j-1} + I_j \circ A_j \\ H_j = O_j \circ (\tanh C_j) \end{array} \right. \tag{2}$$

4.1. φ-stress operators

As indicated in Equation 2, we can stress an LSTM cell in its weights using variants of φ stress operators or functions with different ρ stress intensities and φ_ρ stressing operator (see Equation 3).

$$\varphi(\rho, x) = \begin{cases} \varphi_\rho(x) & \varphi \quad \textit{activated} \\ x & \varphi \quad \textit{not activated} \end{cases} \tag{3}$$

For weight stressing, we have adopted Gaussian stress φ_g and deep Gaussian stress φ_f functions. The gaussian stress φ_g (see Algorithm 1) is defined as a gaussian random variable G_X (i.e. x represents the observed values of variable G_X), where ρ is the standard deviation and x is the mean. So, the stressing range of a given input value x is the interval $[x-3\rho, x+3\rho]$ to cover 99% of the random stressing distribution. For each value of x, a set of Gaussian stress functions φ_k are defined on the neighborhood of x and are characterized by their respective parameters ρ_k. Thus, the question is to determine the optimal stress intensity $\rho^\star$ of $\{\rho_k, k \in \mathbb{N}\}$ which produces adequate stress for control of Deep LSTM model robustness.

Let us consider the optimum $\rho^\star$ with a better dispersion impact on the most sensitive weierstrass function $W_{(a,b)}$ [26] or determined by using an adequate optimization algorithm. In this context, we pointed out the perspective of the use of an adaptive ρ stress thresholding.

Algorithm 1: $\varphi_g(\rho^\star,\ x) : x'$

Data: $\rho^\star$: Optimal intensity threshold of stressing. % ρ is the stressing ratio;
x : input value to stress;
Result: x' : stressed value;

1 Initialization;
2 $r \leftarrow G_X()$ % random Gaussian variable ;
3 $x' \leftarrow (((x - 3 * \rho^\star) + (6 * \rho^\star) * r))$;
4 **return** x';

Next, we proposed a deep gaussian stress φ_f function, introduced in Algorithm 2, where we consider floating-scaled low-intensity stresses. The stress variation for an input value x is considered in *ulp* (unit in Last Place). Thus, optimal $\rho^\star$ parameter is used as a jump of floor exposant value e of the input x (see Algorithm 2, instructions $4, 5$).

Algorithm 2: $\varphi_f(\rho^\star,\ x) : x'$

Data: $\rho^\star$: Optimal intensity threshold of stressing. % $\rho^\star$ is the stressing ratio;
x : input value to stress;
Result: x' : stressed value;

1 Initialization;
2 $r \leftarrow Gaussian_X()$ % random Gaussian variable ;
3 e : integer % exponent range of the Unit in Last Place (Ulp) ;
4 $e \leftarrow floor(|log10(Ulp(x)) * \rho^\star)|)$;
5 $\sigma \leftarrow 10^e * |(x - nextFloat(x, Up))|$;
6 $x' \leftarrow (((x - 3 * \sigma) + (6 * \sigma) * r))$;
7 **return** x';

After implementation of these two variants of φ-stress functions, the key issue is the choice of good stress φ functions and the tuning of parameter ρ, to have adequate model-level based mutants.

4.2. Fuzzy Deep LSTM Model

It is well known that the internal logic of Deep LSTM model M is composed of multiple layers $L_{i,i=1\cdots k}$. Each layer is composed of a set of LSTM cells and followed at the final stage by a fully connected feed-forward neural networks (FNN). Toese LSTM cells within the LSTM layers [8] have the same structure with different weight parameter values. The aim here is to construct mutant models from a pre-trained Deep LSTM model M.

The concept is to derive the fuzzy weights of the original Deep LSTM pre-trained model M. The Fuzzing is done holistically on the different layers and cells of M (see Figures 5 and 6). As indicated previously, we have adopted a model-level mutating process on a given Deep LSTM pre-trained model M. The mutant models M' were generated by using φ-stress operators or functions. These stress functions have different scales of intensities ρ determined and validated on the Weierstrass function [26].

We notice that φ-stress is applied on component nodes A (input), I (input gate), O (output gate), and F (Forgate gate) of a given Deep LSTM model M (see dashed components in Figure 6). A ρ-stress intensity impacted each layer $L_{i=1\cdots 4}$ and its LSTM cell weights $\{w_A, w'_A, w_I, w'_I, w_O, w'_O, w_F, w'_F\}$.

$$\text{Algebraic Fuzzy Model for LSTM cell} \left| \begin{array}{l} I_j = \sigma(t_j \times \varphi(w_I) + H_{j-1} \times \varphi(w'_I) + b_I) \\ O_j = \sigma(t_j \times \varphi(w_O) + H_{j-1} \times \varphi(w'_O) + b_O) \\ F_j = \sigma(t_j \times \varphi(w_F) + H_{j-1} \times \varphi(w'_F) + b_F) \\ A_j = \tanh\left(t_j \times \varphi(w_A) + H_{j-1} \times \varphi(w'_A) + b_A\right) \\ C_j = F_j \circ C_{j-1} + I_j \circ A_j \\ H_j = O_j \circ (\tanh C_j) \end{array} \right. \tag{4}$$

Consequently, different fuzzy deep mutant models M' could be constructed with respect to the Equation 4. A set L of generated mutant model $M^{'}$ modeled by $N_L(M, \varphi_f, \rho, \epsilon)$ would be used to test the robustness of the model M (see the Figure 7).

In addition, robustness is determined with respect to a tolerance range $\epsilon \in [0, 1]$ which expresses that a model M is robust in regards to a ϵ tolerance. The issue is to analyze performance improvement of original Deep LSTM Model M in relation to φ-stress intensity threshold ρ and for a robustness tolerance region.

4.3. φ-Stress Dispersion Tuning and Validation

We now consider an original pre-trained Deep LSTM model M and a set of its model-level mutants M', which is generated by a given φ stressing operator. We notice that robustness testing efficiency of the model M depends on the quality of generated mutant models M'. Consequently, the choice of φ stress function and its intensity ρ are crucial for the quality of our robustness framework.

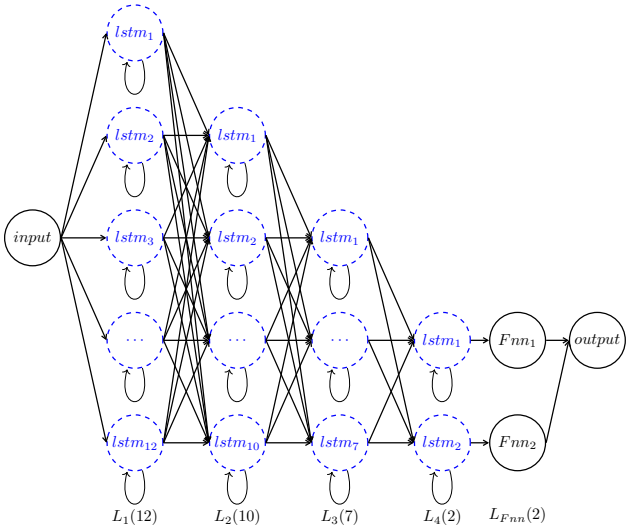

Figure 5: Fuzzy Deep LSTM-NN Network

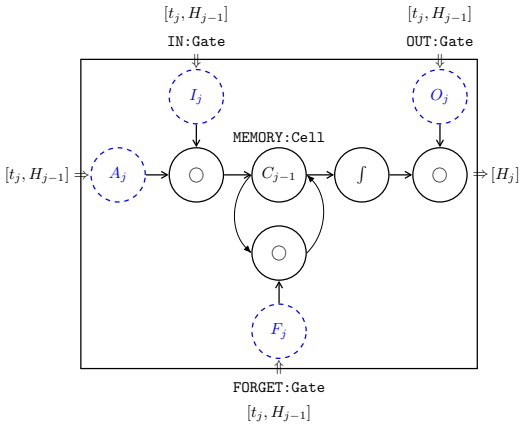

Figure 6: Fuzzy LSTM cell

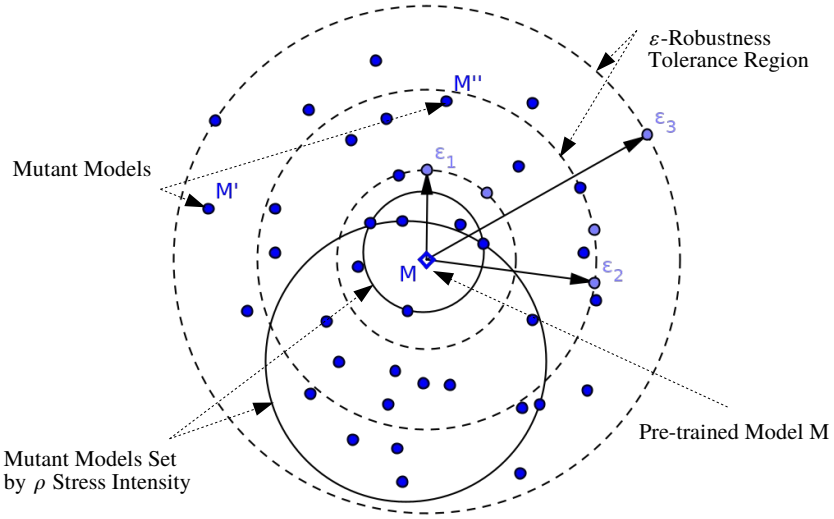

Figure 7: Mutant models $M' = N_L(M, \varphi, \rho, \epsilon)$ for pre-trained Deep Model M.

So, to tune correctly the ρ-threshold stress parameter, we have evaluated the dispersion index of gaussian φ_g and deep gaussian φ_f stressing operators. Furthermore, to strengthen and to refine the quality of φ-stress functions one could also adopt learning techniques with their k-fold cross-validation procedure to evaluate ρ hyperparameter for the proposed φ_ρ-stress.

In our proposed robustness testing framework for RUL prediction Deep LSTM models, the Weierstrass function $W_{(a,b)}$ [26] was defined as a stress reference function with respect

to its sensitivity. Besides, for φ stress tuning and validation a set of ($\times$10000) values was considered from Weierstrass function $W_{(a,b)}$. This latter ensures a formal statistical hypothesis on the degree of randomness for different ρ stressing values. Also, the scale variation hypothesis is considered to describe the stress φ functions and a scale deviation with its corresponding index dispersion characterizes the parameter ρ for the corresponding φ stress operator.

After varying ρ in the interval $[0, 1]$ and applying the stress operators φ on n values set of Weierstrass function, one can estimate the dispersion index (σ^2/μ) of the observed distribution variation between the original Weierstrass set values $\{W_{(a,b)}(x_i) \mid i \in [1,\ n]\}$ and their corresponding φ-stressed values $\{\varphi(\rho,\ W_{(a,b)}(x_i)) \mid i \in [1,\ n]\}$ in respect to a given hyperparameter threshold ρ (see Table 3 and Table 4).

ρ-**threshold**	σ-**stdeviation**	*scale*-**stdeviation**	μ-**mean**	Index σ^2/μ
0.05	0.000001	1.0×10^{-6}	0.0000003	0.0000033
0.10	0.0000009	9.0×10^{-7}	0.0000003	0.0000027
0.15	0.0000078	7.8×10^{-6}	0.0000039	0.0000156
0.20	0.0000072	7.2×10^{-6}	0.0000029	0.0000179
0.25	0.0000259	2.59×10^{-5}	0.0000092	0.0000729
0.30	0.0000813	8.13×10^{-5}	0.0000374	0.0001767
0.35	0.0000916	9.16×10^{-5}	0.0000411	0.0002041
0.40	0.0002318	2.318×10^{-4}	0.000103	0.0005217
0.45	0.0007187	7.187×10^{-4}	0.0003145	0.0016424
0.50	0.0025114	2.5114×10^{-3}	0.0011536	0.0054673
0.55	0.0026653	2.6653×10^{-3}	0.0010538	0.0067412
0.60	**0.0067376**	$6.7376 \times \mathbf{10^{-3}}$	0.0028339	**0.0160187**
0.65	0.0241251	2.41251×10^{-2}	0.0070331	0.0827545
0.70	**0.027344**	$2.7344 \times \mathbf{10^{-2}}$	0.0082145	**0.0910213**
0.75	**0.2508882**	$2.508882 \times \mathbf{10^{-1}}$	0.0903887	**0.6963801**
0.80	0.2548509	2.548509×10^{-1}	0.0900848	0.720976
0.85	0.2332845	2.332845×10^{-1}	0.0889307	0.6119558
0.90	**2.6054599**	$2.6054599 \times \mathbf{10^{0}}$	0.9998489	**6.7894472**
0.95	2.087747	2.087747×10^{0}	0.7539066	5.7814689
1.0	23.983631	$2.3983631 \times 10^{+1}$	8.3560683	68.837943

Table 3: The ρ-threshold parameter tuning, for deep stress φ_f

When the dispersion index (σ^2/μ) is equal to 1, the stress is considered as a poisson process and the mutant models $M^{'}$ are *well-dispersed*. Once more, if the dispersion index is less than 1, the constructed mutant models $M^{'}$ are defined as *under-dispersed*. This indicates that the stress function is *regular* and *periodic* and the mutant models $M^{'}$ are really near from the original model M. In this case, difference pattern measures of Weierstrass set of values with respect to their stressed values are under-dispersed and the stress will not be conclusive. Even more, for an index dispersion larger than 1, the difference suite set (original values - stressed values) is *over-dispersed*, and mutant models $M^{'}$ are highly diversified and

ρ-**threshold**	σ-**stdeviation**	*scale*-**stdeviation**	μ-**mean**	Index σ^2/μ
0.05	**0.3108847**	3.108847×10^{-1}	0.1372421	**0.704225**
0.10	0.6329393	6.329393×10^{-1}	0.3293736	1.2162848
0.15	0.8996437	8.996437×10^{-1}	0.4576311	1.7685832
0.20	1.2344517	1.2344517×10^{0}	0.5150154	2.9588841
0.25	1.5386008	1.5386008×10^{0}	0.5985575	3.954996
0.30	1.925791	1.925791×10^{0}	1.099937	3.3717124
0.35	**2.0713309**	2.0713309×10^{0}	0.9726125	**4.4112239**
0.40	2.30051	2.30051×10^{0}	0.9484754	5.5798456
0.45	2.8412925	2.8412925×10^{0}	1.2642861	6.3853769
0.50	3.051444	3.051444×10^{0}	1.5096636	6.1678048
0.55	3.1224808	3.1224808×10^{0}	1.453115	6.7096454
0.60	**3.9597845**	3.9597845×10^{0}	1.9854777	**7.8972901**
0.65	3.9870738	3.9870738×10^{0}	1.9026672	8.3549858
0.70	4.1758926	4.1758926×10^{0}	2.1342777	8.1704828
0.75	4.4210885	4.4210885×10^{0}	2.2468422	8.699331
0.80	4.7083735	4.7083735×10^{0}	2.4729038	8.964676
0.85	5.4860879	5.4860879×10^{0}	2.9822877	10.091971
0.90	4.9955571	4.9955571×10^{0}	3.2179195	7.7551942
0.95	5.7202667	5.7202667×10^{0}	2.5832466	12.666794
1.0	5.9164803	5.9164803×10^{0}	3.6374172	9.6235148

Table 4: The ρ-threshold parameter tuning, for Gaussian stress φ_g.

are not good for the model M robustness testing. This is due to the fact that the generated mutant models M' are in a widely range dispersion.

From Tables 3 and 4, it can be noticed that gaussian stress function φ_g goes quickly to an *overdispersion* of the mutant models M' when ρ exceeds 0.05. In contrast, deep stress function φ_f is better in terms of smooth stressing, and it covers all stress categories *under*, *well*, and *over* dispersions. Considering the two stress functions φ_f and φ_g, we have retained for our experiments, the ρ values introduced in Table 5.

Indeed, the threshold $\rho = 0.60$ and $\rho = 0.70$ for the φ_f stress function cover the range ρ variation in $[0.05, 0.75[$ and represent an under dispersed generation of mutant models M' (see Table 3). The value $\rho = 0.75$ is defined for the threshold range $[0.75, 0.90[$ and characterize the well-dispersed mutant models. Finally, when $\rho \in [0.90, 1.0]$, the models M' are overdispersed (see index dispersion in Table 3). However, for the more sensitive φ_g, we have chosen two ρ parameter values. First, $\rho = 0.05$ for nearest well-dispersed mutant models and second, $\rho \in \{0.35, 0.60\}$ for far overdispersed models M'. In Table 5, we can see clearly that when a dispersion index between Weierstrass values and theirs φ-stressed ones is in the neighborhood of 1, the mutant models M' are considered as well-dispersed and so, useful for the robustness testing of the model M.

φ-**Stress**	ρ-**threshold**	σ-**stdeviation**	*scale*-**stdeviation**	μ-**mean**	Index σ^2/μ
Deep Gaussian Stress φ_f	—	—	—	—	—
φ_f	**0.60**	0.0067376	$6.7376 \times \mathbf{10^{-3}}$	0.0028339	**0.0160187**
φ_f	**0.70**	0.027344	$2.7344 \times \mathbf{10^{-2}}$	0.0082145	**0.0910213**
φ_f	**0.75**	0.2508882	$2.508882 \times \mathbf{10^{-1}}$	0.0903887	**0.6963801**
φ_f	**0.90**	2.6054599	$2.6054599 \times \mathbf{10^{0}}$	0.9998489	**6.7894472**
Gaussian Stress φ_g	—	—	—	—	—
φ_g	**0.05**	0.3108847	$3.108847 \times \mathbf{10^{-1}}$	0.1372421	**0.704225**
φ_g	**0.35**	2.0713309	$2.0713309 \times \mathbf{10^{0}}$	0.9726125	**4.4112239**
φ_g	**0.60**	3.9597845	$3.9597845 \times \mathbf{10^{0}}$	1.9854777	**7.8972901**

Table 5: The ρ-threshold parameter tuning for stress φ.

5. ϵ-Robustness for Deep LSTM model

Several Deep LSTM RUL prediction models were published in [30, 31, 32, 33] to provide their competitive positions in terms of correctness (i.e. RMSE, Equation 5) and quality (i.e. Scoring with an error E, Equation 6). Thus, the smaller are RMSE and Scoring [8, 34, 24], the better are result and precision of the proposed RUL prediction model.

$$\mathcal{R} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} {E_i}^2} \tag{5}$$

$$\mathrm{S} = \begin{vmatrix} \sum_{n}^{i=1} (e^{\frac{-Ei}{13}} - 1) \text{ for } (E_i < 0) \\ \sum_{n}^{i=1} (e^{\frac{-Ei}{10}} - 1) \text{ for } (E_i >= 0) \end{vmatrix} \tag{6}$$

Indeed, RUL prediction metrics enable to find best weights $w_I, w_O, w_F, w_A,\ {w'}_I, {w'}_O, {w'}_F, {w'}_A$ and biases b_I, b_O, b_F, b_A for minimizing the $\mathcal{R}ul$ function defined in Equation 7 (i.e. the variation between real remaining useful life ${R_j}^{calc}$ and computed value ${R_j}^{est}$, in a state j).

$$\mathcal{R}ul = \sum_{j} ({R_j}^{est} - {R_j}^{calc})^2 \tag{7}$$

The performance robustness of a Deep LSTM model M is defined with respect to $RMSE$ metric $\mathcal{R}_M$ and Scoring $\mathcal{S}_M$. Thus, each mutant model M' associated with the original model M has a performance determined by ratio deviations introduced in Equations 8 and 9. These ratio metrics determine the deviation of the mutant model M' from the original model M. When these ratios are zero, the models M and M' are similar. Here, the maximum performance deviation between M' and M was considered as equal to 1 (i.e. this represents the maximum deviation of M' from the original model M in terms of RMSE and scoring measurements).

$$R_{\circledS} = \left| \frac{\mathcal{R}_{\mathcal{M}'} - \mathcal{R}_{\mathcal{M}}}{\mathcal{R}_M} \right| \tag{8}$$

$$S_{\circledS} = \left| \frac{\mathcal{S}_{\mathcal{M}'} - \mathcal{S}_{\mathcal{M}}}{\mathcal{S}_M} \right| \tag{9}$$

To integrate correctness $R_{\circledS}$ and quality $S_{\circledS}$ deviations of a mutant M' from a φ-stressed Deep LSTM model M, we define the robustness of M as a linear evolution expression of its mutant models (Proposition 1).

Proposition 1 (Robustness$^{\alpha}$). *A Robustness for a model M is defined as the combination of the shift performance $S_{\circledS}$ and $R_{\circledS}$ of its mutant models M', with respect to the characterized α parameter.*

$$Robust^{\alpha} = \ (1-\alpha) * R_{\circledS} + \alpha * S_{\circledS} \quad , \alpha \in [0,1], \forall M' \tag{10}$$

We notice that the parameter α is used to measure the combined impact of scoring sensitivity (*Quality*) and the RMSE specificity (*Correctness*) of the mutant models M'. If the shift performances are impacted in the same manner, the formula in Equation (10) will be $(R_{\circledS} + S_{\circledS})/2$ and the parameter α is used to be 0.5.

Additionally, an ϵ-Robustness of a model M was introduced as the tolerance performance response of M with respect to φ-stress function and its ρ intensity stressing on M (Proposition 2).

Proposition 2 (ϵ-robustness). *Let ϵ be a given value in $[0,1]$. An ϵ-robustness for a model M is defined as the robustness ${Robust_{\epsilon}}^{\alpha}$ of the mutant models M' for the ϵ parameter.*

$${Robust_{\epsilon}}^{\alpha} = \begin{cases} 1 & if \quad (1-\alpha) * R_{\circledS} + \alpha * S_{\circledS} \leq \epsilon \\ 0 & else \end{cases} \tag{11}$$

Consequently, robustness and ϵ-robustness of a model M were considered as a combined measure relative to the ϵ parameter [35, 36]. This means that when ϵ tends to 1, the robustness range of the model M has a large tolerance to the stress and many possibilities of robust deep models can be constructed.

Otherwise, when the ϵ comes close to zero, the robustness range of M is sensitive to the φ-stresses, and limited ϵ-robustness models are considered (see Algorithm 3). The ϵ-robustness for M is defined in the interval range $[0,1]$ as the mean of the shift performance metrics of the mutants M'.

Algorithm 3: ROBUSTNESS$_\epsilon(T, T', W) : \{true, false\}$

Data: T : Data sets training;
T' : Data sets Testing;
W : Random suite Weierstrass set values;
Result: true/false;

```
% Initialisation
Let given ϵ;
Generation of Deep LSTM Network M using T;
Construction of φ-stress operator using W;
% initial ρ_0 intensity and its dispersion index by regards to W;
ρ ← ρ_0;
% Tuning stress ρ intensity for a given φ operator;
while (ρ is not eligible) do;
ρ ← ρ';
Computing of the new ρ and its dispersion index;
endwhile
Mutants M' generation of the Network Model M using Fuzzing φ-Operator and
  intensity ρ;
% Robustness Testing
Computation of performance metrics R_Ⓢ and S_Ⓢ on data sets testing T';
ϵ-Robustness testing of M for data sets testing T';
if (M is ϵ-Robust) then return true;
else return false;
```

6. Experiments

In our experiments, we suppose that under a normal and healthy condition states, the engine's life decreases progressively until the occurrence of a failure or a run-to-failure event. The goal is to predict the remaining useful cycles (RUL) of a system component and deduce cycles numbers before a raw data failure, which allows performing maintenance in time and to avoid any crashing.

6.1. Data Sets

We have used for the validation of our proposed framework, the C-MAPSS - *Commercial Modular Aero-Propulsion System Simulation* - data sets [37]. These data sets were considered as well characterizing fault evolution since the C-MAPSS experiment simulations are carried out under different combinations of operational conditions and failure modes (see Table 6). More specifically, several sensor channels are used to track the wear evolution of a turbofan engine degradation. The generated data sets are organized in four data subsets with 26 criteria and many data rows, where each row includes a unit engine identity (id) and a current working cycle number $Ncycle$.

Data Sets (size)	**fd001**	**fd002**	**fd003**	**fd004**
Training Data-set	20631	53759	24720	61249
Testing Data-set	13096	33991	16596	41214
Min. cycles to failure (TR)	128	128	145	128
Max. cycles to failure (TS)	362	378	525	543
Min. cycles with no failure (TS)	31	21	38	19
Max. cycles with no failure (TS)	303	267	475	486
#Engines (TR)	100	260	249	249
#Engines (TS)	100	259	100	248
Oper. conditions	1	6	1	6
Fault conditions	1	1	2	2

Table 6: C-MAPSS : Training(TR) and Testing(TS) data sets

6.2. Experimental guides

Let us consider Deep LSTM model M for RUL prediction, characterized by its RMSE R_M and scoring S_M. We have constructed for M, a set of mutant models M' generated by applying different intensities ρ of the φ-stress functions introduced previously in Section 4. The stresses affect the LSTM cells weights $\{w_A, w'_A, w_I, w'_I, w_O, w'_O, w_F, w'_F\}$ in a holistic manner, under the condition that an LSTM cell component (ie. INPUT, IN-Gate, OUT-Gate, and FORGET-Gate functions) should not be stressed on their input and output at the same time.

Furthermore, to carry out robustness testing, stress intensity ρ, and robustness tolerance ϵ parameters were tuned for index dispersion of the φ-stress (see Table 5). Therefore, ρ is setted to $\{0.60, 0.70, 0.75, 0.90\}$ for deep gaussian φ_f, and to $\{0.05, 0.35, 0.60\}$ for gaussian φ_g function. Concerning the tolerance robustness parameter ϵ, the thresholds were fixed respectively to $\{0.05, 0.50, 0.90\}$ for *minimal*, *medium*, and *maximal*.

Throughout our experiments, a mutant Deep LSTM network for RUL prediction M' is characterized by $M' = N(M, \varphi, \rho, \epsilon)$, where φ is the stress operator, ρ the stress intensity, and ϵ the robustness tolerance of the model M. For this original model M, we considered 4 layers, and 4 cell LSTM components for each layer, with seven 7 φ-stress intensities ρ and 3 categories of tolerance robustness. Thus, the total mutant models M' for our experiments is $|4| \times |4| \times |7| \times |3| = 336$ mutant models.

6.3. Results and Discussion

In this stage, we consider the hyper-parameters ϵ and ρ for robustness testing of a pre-trained deep model M using its 336 mutant models M'. Firstly, we considered the tolerance of robustness threshold ϵ in its different tolerance levels: minimal (0.05), medium (0.50) and maximal (0.90), independently of parameters ρ, layers L, and weights W. We notice,

in Table 7, that Deep LSTM model M robustness is linearly proportional to the tolerance ϵ over all data sets. It can be seen for $df00x$ that intrinsic robustness criterion of a Deep LSTM network is not covered with only ϵ tolerance parameter.

$N(M, \varphi, \rho, \epsilon)$	ϵ-Robustness *fd001 (%)*	ϵ-Robustness *fd002 (%)*	ϵ-Robustness *fd003 (%)*	ϵ-Robustness *fd004 (%)*	ϵ-Robustness *fd00x* (%)
$N(M, \varphi, \rho,$ **0.15**$)$	43	41	37	50	**43**
$N(M, \varphi, \rho,$ **0.50**$)$	49	48	46	56	**50**
$N(M, \varphi, \rho,$ **0.90**$)$	54	53	50	63	**55**

Table 7: ϵ-robustness percentage of 336 mutants

Secondly, we studied the ϵ-robustness of model M using the mutant networks over intensity ρ variation for the stress operators φ_f and φ_g (see Table 8). It is clear that when the stress intensity ρ grows the robustness of the model M tends to be reduced. Also, the φ_f seemed to be better for our framework, since the stress φ_g function is more sensitive (see bold values in Table 8).

$N(M, \varphi, \rho, \epsilon)$	ϵ-Robustness (%) $\epsilon_{minimal} = 0.15$	ϵ-Robustness (%) $\epsilon_{medium} = 0.50$	ϵ-Robustness (%) $\epsilon_{maximal} = 0.90$
Deep Gaussian Stress φ_f	—	—	—
$N(M, \varphi_f, \rho = 0.60, \epsilon)$	99	99	99
$N(M, \varphi_f, \rho = 0.70, \epsilon)$	95	93	88
$N(M, \varphi_f, \rho = 0.75, \epsilon)$	80	72	59
$N(M, \varphi_f, \rho = 0.90, \epsilon)$	**20**	**15**	**07**
Gaussian Stress φ_g	—	—	—
$N(M, \varphi_g, \rho = 0.05, \epsilon)$	48	43	34
$N(M, \varphi_g, \rho = 0.35, \epsilon)$	**26**	**16**	**07**
$N(M, \varphi_g, \rho = 0.60, \epsilon)$	**18**	**09**	**04**

Table 8: ϵ-robustness percentage over ρ variation

Finally, we chose φ_f stress function with the associated ρ intensities values $\{0.60, 0.70, 0.75\}$ to discuss the internal logic LSTM model layers and weights stressing for the model M. This is due to the fact that φ_f is adequate for our robustness testing framework and some generated mutant models M' have better performance than M, in terms of correctness (RMSE) and quality (Score). The impact of the $\rho \in \{0.60, 0.70, 0.75\}$ variation on the different layers $L_{i=1\cdots4}$, indicates that layers are more impacted when stress intensity increases, and also layer 1 seemed to be more sensitive with regards to the others layers (see Tables 9, 10 and 11).

$N(M, \varphi, \rho, \epsilon = 0.15)$	Layer $\#L$	ϵ-Robustness *fd001 (%)*	ϵ-Robustness *fd002 (%)*	ϵ-Robustness *fd003 (%)*	ϵ-Robustness *fd004 (%)*
Stress Intensity $\rho = 0.60$	—	—	—	—	—
$N(M, \varphi_f, 0.60, \epsilon)$	1	100	100	100	**86**
$N(M, \varphi_f, 0.60, \epsilon)$	2,3,4	100	100	100	100
Stress Intensity $\rho = 0.70$	—	—	—	—	—
$N(M, \varphi_f, \mathbf{0.70}, \epsilon)$	1	**63**	**75**	**75**	**75**
$N(M, \varphi_f, 0.70, \epsilon)$	2	100	100	63	100
$N(M, \varphi_f, 0.70, \epsilon)$	3	100	100	63	100
$N(M, \varphi_f, 0.70, \epsilon)$	4	100	100	100	100
Stress Intensity $\rho = 0.75$	—	—	—	—	—
$N(M, \varphi_f, \mathbf{0.75}, \epsilon)$	1	**25**	**38**	**38**	**25**
$N(M, \varphi_f, 0.75, \epsilon)$	2	86	75	38	75
$N(M, \varphi_f, 0.75, \epsilon)$	3	75	75	0.0	88
$N(M, \varphi_f, 0.75, \epsilon)$	4	75	88	75	75

Table 9: ϵ-robustness percentage over the layers for $\epsilon = 0.15$

$N(M, \varphi, \rho, \epsilon = 0.50)$	Layer $\#L$	ϵ-Robustness *fd001 (%)*	ϵ-Robustness *fd002 (%)*	ϵ-Robustness *fd003 (%)*	ϵ-Robustness *fd004 (%)*
Stress Intensity $\rho = 0.60$	—	—	—	—	—
$N(M, \varphi_f, \mathbf{0.60}, \epsilon)$	1	100.0	100.0	100.0	**87.5**
$N(M, \varphi_f, 0.60, \epsilon)$	2,3,4	100.0	100.0	100.0	100.0
Stress Intensity $\rho = 0.70$	—	—	—	—	—
$N(M, \varphi_f, \mathbf{0.70}, \epsilon)$	1	**87.5**	**75.0**	**87.5**	**75.0**
$N(M, \varphi_f, 0.70, \epsilon)$	2	100.0	100.0	87.5	100.0
$N(M, \varphi_f, 0.70, \epsilon)$	3	100.0	100.0	75.0	100.0
$N(M, \varphi_f, 0.70, \epsilon)$	4	100.0	100.0	100.0	100.0
Stress Intensity $\rho = 0.75$	—	—	—	—	—
$N(M, \varphi_f, \mathbf{0.75}, \epsilon)$	1	**37.5**	**62.5**	62.5	**37.5**
$N(M, \varphi_f, 0.75, \epsilon)$	2	100.0	75.0	**37.5**	87.5
$N(M, \varphi_f, 0.75, \epsilon)$	3	75.0	100.0	37.5	87.5
$N(M, \varphi_f, 0.75, \epsilon)$	4	87.5	87.5	87.5	87.5

Table 10: ϵ-robustness percentage over the layers for $\epsilon = 0.50$

Now, we discuss the ϵ-robustness of the model M for a φ-stress intensity $\rho = 0.75$ over the variation of ϵ. The percentage of the mutant models M' retained for the robustness of the pre-trained deep model M is exposed in more detail in Figure 12. The maximum (respectively minimum) percent of tolerant models by type of data-set indicates that $fd003$ is more impacted by the φ-stress operators.

For the stressed weights in Table 13, it seems that INPUT function node A is more sensitive than the others. Also, the IN-Gate, OUT-Gate, and FORGET-Gate functions respond in the same way for a given ρ stressing.

$N(M, \varphi, \rho, \epsilon = 0.90)$	Layer $\#L$	ϵ-Robustness *fd001 (%)*	ϵ-Robustness *fd002 (%)*	ϵ-Robustness *fd003 (%)*	ϵ-Robustness *fd004 (%)*
Stress Intensity $\rho = 0.60$	—	—	—	—	—
$N(M, \varphi_f, \mathbf{0.60}, \epsilon)$	1	100.0	100.0	100.0	**87.5**
$N(M, \varphi_f, 0.60, \epsilon)$	2,3,4	100.0	100.0	100.0	100.0
Stress Intensity $\rho = 0.70$	—	—	—	—	—
$N(M, \varphi_f, \mathbf{0.70}, \epsilon)$	1	**87.5**	**75.0**	**87.5**	**87.5**
$N(M, \varphi_f, 0.70, \epsilon)$	2	100.0	100.0	100.0	100.0
$N(M, \varphi_f, 0.70, \epsilon)$	3	100.0	100.0	87.5	100.0
$N(M, \varphi_f, 0.70, \epsilon)$	4	100.0	100.0	100.0	100.0
Stress Intensity $\rho = 0.75$	—	—	—	—	—
$N(M, \varphi_f, \mathbf{0.75}, \epsilon)$	1	**62.5**	**62.5**	75.0	**50.0**
$N(M, \varphi_f, 0.75, \epsilon)$	2	100.0	87.5	75.0	100.0
$N(M, \varphi_f, 0.75, \epsilon)$	3	75.0	100.0	**50.0**	87.5
$N(M, \varphi_f, 0.75, \epsilon)$	4	87.5	87.5	87.5	87.5

Table 11: ϵ-robustness percentage over the layers for $\epsilon = 0.90$

$N(M, \varphi, \rho = 0.75, \epsilon)$	Layer $\#L$	ϵ-Robustness *fd001 (%)*	ϵ-Robustness *fd002 (%)*	ϵ-Robustness *fd003 (%)*	ϵ-Robustness *fd004 (%)*
Robustness tolerance $\epsilon = 0.15$	—	—	—	—	—
$N(M, \varphi_f, \rho, 0.15)$	1	**25**	**38**	38	**25**
$N(M, \varphi_f, \rho, 0.15)$	2	86	75	38	75
$N(M, \varphi_f, \rho, 0.15)$	3	75	75	**0.0**	88
$N(M, \varphi_f, \rho, 0.15)$	4	75	88	75	75
Robustness tolerance $\epsilon = 0.50$	—	—	—	—	—
$N(M, \varphi_f, \rho, 0.50)$	1	37.5	62.5	62.5	37.5
$N(M, \varphi_f, \rho, 0.50)$	2	**100.0**	75.0	**37.5**	87.5
$N(M, \varphi_f, \rho, 0.50)$	3	75.0	**100.0**	**37.5**	87.5
$N(M, \varphi_f, \rho, 0.50)$	4	87.5	87.5	87.5	87.5
Robustness tolerance $\epsilon = 0.90$	—	—	—	—	—
$N(M, \varphi_f, \rho, 0.90)$	1	62.5	62.5	75.0	50.0
$N(M, \varphi_f, \rho, 0.90)$	2	**100.0**	87.5	75.0	**100.0**
$N(M, \varphi_f, \rho, 0.90)$	3	75.0	**100.0**	50.0	87.5
$N(M, \varphi_f, \rho, 0.90)$	4	87.5	87.5	87.5	87.5

Table 12: ϵ-robustness percentage over the layers for ϵ

$N(M, \varphi, \rho, \epsilon = 0.90)$	$\#w$	*fd001 (%)*	*fd002 (%)*	*fd003 (%)*	*fd004 (%)*	fd00x (%)
Weights for Node A	—	—	—	—	—	—
$N(M, \varphi, \rho, \epsilon)$	w_A	**28.57**	**35.71**	**25.0**	**35.71**	**31.25**
$N(M, \varphi, \rho, \epsilon)$	${w'}_A$	**39.29**	**35.71**	**32.14**	**39.29**	**36.61**
Weights for Node I	—	—	—	—	—	—
$N(M, \varphi, \rho, \epsilon)$	w_I	60.71	67.86	50.0	75.00	63.39
$N(M, \varphi, \rho, \epsilon)$	${w'}_I$	64.29	60.71	64.29	75.00	66.07
Weights for Node O	—	—	—	—	—	—
$N(M, \varphi, \rho, \epsilon)$	w_O	57.14	57.14	53.57	67.86	58.93
$N(M, \varphi, \rho, \epsilon)$	${w'}_O$	60.71	57.14	67.86	71.43	64.29
Weights for Node F	—	—	—	—	—	—
$N(M, \varphi, \rho, \epsilon)$	w_F	57.14	57.14	57.14	64.29	58.93
$N(M, \varphi, \rho, \epsilon)$	${w'}_F$	64.29	50.0	53.57	75.00	60.72

Table 13: ϵ-robustness percentage over ρ for LSTM cell weights and $\epsilon = 0.90$

7. Conclusion

In this paper, we have developed a robustness testing framework for RUL prediction Deep LSTM model M. The proposed framework introduces also ϵ-robustness testing for RUL prediction Deep LSTM models. The metrics and measures used in our framework are RMSE and scoring RUL prediction estimates. We adopted a model-level mutation technique to generate mutant model M' for a pre-trained model M. This is highly desirable for less consuming time with regards to the source-level and program-level mutations where we have to train the mutant models from scratch zero. The stressing φ-functions in the robustness testing framework, were defined and validated using the Weierstrass function known to be continuous everywhere but differentiable nowhere. Then, φ-stress operators were holistically applied on the model M, impacting the weights of the LSTM cells and layers of the RUL prediction deep learning network. It should be noted that in our framework a better RMSE accuracy and scoring quality values for a deep LSTM network for RUL prediction were ensured, as well as an adequate ε-robustness performance to avoid having Deep LSTM models closely related to the training/testing data sets. The proposed robustness testing framework enables to improve quality evaluation, to reinforce confidence, and to build explainable data-independent Deep LSTM Models for RUL prediction.

The results of our experiments indicate the relevance of the proposed framework in estimating the reliability of layers and weights for RUL prediction Deep LSTM Network. Moreover, tuning of stress ρ and ϵ hyper-parameters should be done adequately to respond to the robustness quality of a specific LSTM deep model for RUL prediction. The issue is to enhance and/or adapt our framework to different domain expertise, having into account the data knowledge vision. In perspective, many questions related to the data are now discussed in the PHM field. Also, we still be concerned to analyze φ-stress operators for new field data sets and to develop new optimization and learning techniques to well estimate ρ intensities and ϵ robustness tolerance for RUL prediction Deep LSTM models.

Acknowledgments

This work has been supported by LITIO laboratory-University Oran1, DGRST and the EIPHI Graduate school (contract "ANR-17-EURE-0002").

References

[1] R. Gouriveau, K. Medjaher, N. Zerhouni, From prognostics and health systems management to predictive maintenance 1: monitoring and prognostics, John Wiley & Sons, 2016.

[2] X.-S. Si, W. Wang, C.-H. Hu, D.-H. Zhou, Remaining useful life estimation–a review on the statistical data driven approaches, European journal of operational research 213 (1) (2011) 1–14.

[3] R. K. Mobley, An introduction to predictive maintenance, Elsevier, 2002.

[4] L. Swanson, Linking maintenance strategies to performance, International journal of production economics 70 (3) (2001) 237–244.

[5] Z. Tian, L. Wong, N. Safaei, A neural network approach for remaining useful life prediction utilizing both failure and suspension histories, Mechanical Systems and Signal Processing 24 (5) (2010) 1542–1555.

[6] C. Okoh, R. Roy, J. Mehnen, L. Redding, Overview of remaining useful life prediction techniques in through-life engineering services, Procedia CIRP 16 (2014) 158–163.

[7] X.-S. Si, Z.-X. Zhang, C.-H. Hu, et al., Data-driven remaining useful life prognosis techniques, Springer, 2017.

[8] S. Zheng, K. Ristovski, A. Farahat, C. Gupta, Long short-term memory network for remaining useful life estimation, in: IEEE International Conference on Prognostics and Health Management (ICPHM), IEEE, 2017, pp. 88–95.

[9] T. Sutharssan, S. Stoyanov, C. Bailey, C. Yin, Prognostic and health management for engineering systems: a review of the data-driven approach and algorithms, The Journal of Engineering (7) (2015) 215–222.

[10] 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 7 (26) (2013) 1751 – 1760.

[11] K. Medjaher, N. Zerhouni, Hybrid prognostic method applied to mechatronic systems, International Journal of Advanced Manufacturing Technology 69 (1) (2013) 823 – 834.

[12] M. F. Dacrema, P. Cremonesi, D. Jannach, Are we really making much progress? a worrying analysis of recent neural recommendation approaches, in: Proceedings of the 13th ACM Conference on Recommender Systems, 2019, pp. 101–109.

[13] K. Javed, R. Gouriveau, N. Zerhouni, State of the art and taxonomy of prognostics approaches, trends of prognostics applications and open issues towards maturity at different technology readiness levels, Mechanical Systems and Signal Processing (MSSP) 94 (2017) 214 – 236.

[14] S.-j. Wu, N. Gebraeel, M. A. Lawley, Y. Yih, A neural network integrated decision support system for condition-based optimal predictive maintenance policy, IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 37 (2) (2007) 226–236.

[15] P. Baruah, R. B. Chinnam, Hmms for diagnostics and prognostics in machining processes, International Journal of Production Research 43 (6) (2005) 1275–1293.

[16] A. Graves, G. Wayne, I. Danihelka, Neural turing machines, 2014.

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

[18] J. L. Elman, Finding structure in time, Cognitive science 14 (2) (1990) 179–211.

[19] F. O. Heimes, Recurrent neural networks for remaining useful life estimation, in: International conference on prognostics and health management, IEEE, 2008, pp. 1–6.

[20] S. Zheng, Z.-Y. Shae, X. Zhang, H. Jamjoom, L. Fong, Analysis and modeling of social influence in high performance computing workloads, in: European Conference on Parallel Processing, Springer, 2011, pp. 193–204.

[21] G. E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural computation 18 (7) (2006) 1527–1554.
[22] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.
[23] S. Zheng, A. Vishnu, C. Ding, Accelerating deep learning with shrinkage and recall, in: 22nd International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2016, pp. 963–970.
[24] G. S. Babu, P. Zhao, X.-L. Li, Deep convolutional neural network based regression approach for estimation of remaining useful life, in: International conference on database systems for advanced applications, Springer, 2016, pp. 214–228.
[25] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, et al., Deepmutation: Mutation testing of deep learning systems, in: IEEE 29th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2018, pp. 100–111.
[26] G. H. Hardy, Weierstrass's non-differentiable function, Vol. 17, 1916, pp. 301–325.
[27] X. Li, Q. Ding, J.-Q. Sun, Remaining useful life estimation in prognostics using deep convolution neural networks, Reliability Engineering & System Safety 172 (2018) 1–11.
[28] M. Sayah, D. Guebli, Z. Al Masry, N. Zerhouni, Towards distribution clustering-based deep lstm models for rul prediction, in: Proceedings - Prognostics and System Health Management Conference, PHM 2020, IEEE, 2020.
[29] N. Jussien, G. Rochart, X. Lorca, Choco: an open source java constraint programming library, 2008.
[30] A. L. Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, H. Zhang, Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture, Reliability Engineering & System Safety 183 (2019) 240–251.
[31] J. Wang, G. Wen, S. Yang, Y. Liu, Remaining useful life estimation in prognostics using deep bidirectional lstm neural network, in: Prognostics and System Health Management Conference (PHM-Chongqing), IEEE, 2018, pp. 1037–1042.
[32] J. Li, X. Li, D. He, A directed acyclic graph network combined with cnn and lstm for remaining useful life prediction, IEEE, 2019.
[33] A. Al-Dulaimi, S. Zabihi, A. Asif, A. Mohammadi, A multimodal and hybrid deep neural network model for remaining useful life estimation, Computers in Industry 108 (2019) 186–196.
[34] Y. Liao, L. Zhang, C. Liu, Uncertainty prediction of remaining useful life using long short-term memory network based on bootstrap method, in: IEEE International Conference on Prognostics and Health Management (ICPHM), IEEE, 2018, pp. 1–8.
[35] D. M. Powers, Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, Journal of Machine Learning Technologies 2 (1) (2011) 37–63.
[36] T. Fawcett, Roc graphs: Notes and practical considerations for researchers, Machine learning 31 (1) (2004) 1–38.
[37] A. Saxena, K. Goebel, D. Simon, N. Eklund, Damage propagation modeling for aircraft engine run-to-failure simulation, in: International conference on prognostics and health management, IEEE, 2008, pp. 1–9.