Neural Turing Machines for the Remaining Useful Life estimation problem

Alex Falcon[a,b], Giovanni D'Agostino[b], Oswald Lanz[c], Giorgio Brajnik[b], Carlo Tasso[b], Giuseppe Serra[b]

[a] *Fondazione Bruno Kessler, Via Sommarive, 18, Povo, Trento, Italy 38123,*
[b] *University of Udine, Via delle Scienze, 206, Udine, Italy 33100,*
[c] *Free University of Bozen-Bolzano, Piazza Domenicani, 3, Bolzano, Italy 39100,*

Abstract

Accurately estimating the amount of productive time remaining to a machine before it suffers a fault condition is a fundamental problem, which is commonly found in several contexts (e.g. mechanical systems) and can have great industrial, societal, and safety-related consequences. Recently, deep learning showed promising results and significant improvements towards a solution to the Remaining Useful Life estimation problem. In this paper, the usage of a sequence model called Neural Turing Machine (NTM), which can be seen as a "computer" that uses the available data to learn how to interact with an external memory, is thoroughly explored. In particular, even by using a single NTM as the key feature extraction component, more accurate solutions can be obtained when compared to widely used Long Short-Term Memory-based solutions. Moreover, such an improvement can be obtained while using fewer learnable parameters. The proposed approach is validated using sensor data of aircraft turbofan engines and particle filtration systems, obtaining competitive results to state-of-the-art techniques. Furthermore, the source code is released at https://github.com/aranciokov/NTM-For-RULEstimation to provide a strong baseline for the community, to support reproducibility and faster advancement in this field.

Keywords: remaining useful life estimation, neural turing machines, neural networks, deep learning

Email address: falcon.alex@spes.uniud.it (Alex Falcon)

1. Introduction

One of the most important problems in the Prognostics and Health Management field is called Remaining Useful Life (RUL) estimation which consists in estimating how long it will take for a mechanical device under analysis to reach a situation where the likelihood of a failure is above a given threshold [1]. On the one hand, estimating the RUL precisely and reliably can have a great impact on maintenance-related costs, since it is possible to foresee when a failure will happen and thus plan accordingly the required intervention. On the other hand, failing such an estimation can have not only crucial economic consequences, but also a decrease in the reliability of a brand and may also create life-threatening situations, *e.g.* the disastrous crumbling of the I-35W in Minneapolis, Minnesota [2] or the more recent Morandi Bridge in Italy [3].

Common approaches for the RUL estimation can be divided into model-based and data-driven. Model-based approaches estimate the remaining life by leveraging mathematical or physical models of the degradation phenomena [4, 5]. These methods often require extensive expert knowledge, expensive verification, and for some components it is quite challenging to establish an accurate physical model. Differently from these approaches, data-driven methods rely on the availability of historical sensor data to build a degradation model. When it is not possible to observe multiple instances of each fault mode, for instance in industrial contexts where a fault may create life-threatening situations, these sensor data can be obtained via simulation software, *e.g.* in [6, 7], or experimental rigs, *e.g.* in [8, 9]. Many of the works following the data-driven approach manually design features in both the time and frequency domains and use them to learn a model through self-organizing maps [10], hidden Markov models [11], etc. While statistics play a major role in deciding which features to use and how to combine them effectively, this feature engineering step can be time consuming and may still rely on prior knowledge. Conversely, deep learning makes it possible to automate the feature extraction process and work directly on the raw sensor data. The more successful deep learning-based approaches for this problem leverage sequence models to extract useful features from the sensor measurements and to identify temporal dependencies in the data. In particular, Long Short-

This is a preprint. The final accepted version is available at:
https://doi.org/10.1016/j.compind.2022.103762

Term Memory networks (LSTM) [12] are widely used as the key component to automate the feature extraction process [13, 14, 15, 16, 17].

An interesting neural network architecture which has not been explored until recently for the RUL estimation problem [18, 19] is the Neural Turing Machine (NTM). NTMs [20] are sequence models which, differently from LSTMs, interact with an external memory decoupled from the computation. As shown by Graves et al. [20], this makes it possible to achieve better performance on several algorithmic tasks, including the "associative recall", which resembles sequence modeling and consists in asking the model to recall an item from a list by querying it with the preceding item. This problem shares some similarities to the estimation of the RUL: if the list consisted of sensor measurements and associated RUL values, the current measurements could be used as a query to obtain the associated RUL value. Therefore, NTMs may also provide a more reliable tool than LSTMs for the RUL estimation problem.

In this paper, it is shown that even by using a simple model made of a single NTM and a decoder based on fully-connected layers it is possible to outperform widely adopted LSTM-based models, while also using fewer learnable parameters (28% less) and therefore with a smaller memory footprint. This is empirically validated with multiple experiments on two public datasets, the C-MAPSS dataset [6] and the PHM Society 2020 Data Challenge dataset [9]. Furthermore, the proposed method obtains competitive results with several state-of-the-art architectures which use deeper networks, bidirectional reasoning, or additional pretraining. Therefore, the experimental results show that providing access to an external memory can be beneficial to deal with the task, possibly implying that NTMs can be a better building block to design more complex architectures and to automatically extract features from the available time series.

The major contributions of this work can be summarized as follows:

- A thorough empirical study is performed to explore the usage of NTMs as the main feature extraction component for the Remaining Useful Life estimation problem.

- Consistent empirical evidences are provided to show that NTMs are a powerful and efficient model which outperforms the more popular LSTM-based models, while being also less parameter-demanding and thus more memory-efficient which makes it possible to use in memory-constrained environments. Given the similar underlying nature of the

two sequence models, it is possible to integrate the NTM within other architectures and improve the final performance.

- The proposed simple model achieves an estimation error comparable and even competitive with respect to the error obtained by other architectures found in literature, which are more complex, use ensemble of models, and additional pretraining.
- Multiple experiments are performed on two public datasets dealing with aircraft engines (C-MAPSS) and particle filtration systems (PHM Society 2020 Data Challenge), showing the strengths of the proposed model while also highlighting some limitations related to industrial contexts.
- The source code is released, to ensure reproducibility and to provide a strong, open source baseline otherwise difficultly found in the community.

In Section 2 the scientific literature related to this manuscript is introduced. Then, in Section 3 the details about the methodology are described, by focusing on the application of the Neural Turing Machine to the RUL estimation task. All the experiments performed throughout this manuscript are described in Section 4. Finally, in Section 5 some conclusions are drawn about the work done and possible future works are mentioned.

2. Related Work

Model-based and data-driven methods. Estimating the remaining useful life of a system has been a strategic research problem for several decades [21, 22]. Traditional approaches can be divided into model-based and data-driven.

The former are based on the availability of mathematical or physical models of the degradation phenomena, such as spall propagation models for rolling bearing elements [23] or crack growth models for a system experiencing fatigue [24]. Model-based methods require an in-depth understanding of the underlying system and the failure modes. Furthermore, these approaches are built in a case-by-case scenario, making them difficult to be applied in different contexts without spending a considerable effort.

Methods following a data-driven approach rely on the availability of historical sensor data to build a degradation model. In situations where the

machinery is frequently maintained and faults are never observed, the sensor data can be obtained through simulation models [6, 7] or through experimental rigs [8, 9]. Notable examples of data-driven methods include statistical methods, such as Auto Regressive Integrated Moving Average [25, 26] and hidden Markov models [11, 27], and Artificial Intelligence methods, *e.g.* by using self-organizing maps [10] and Support Vector Machines [28]. Differently from the model-based approach, the data-driven methodology does not require a deep understanding of the underlying system. Yet, in data-driven methods which do not employ deep learning the features are manually designed and extracted from the raw data, therefore this feature engineering step can be time consuming and may still rely on domain knowledge.

Deep Learning-based methods. Differently from previous approaches, deep learning makes it possible to automate the process of feature extraction from the raw sensor data. The attention towards these techniques has been promoted thanks to the availability of public datasets (*e.g.* [6, 8]) and the possibility to exploit high quality sensors to frequently measure the evolution of different characteristics of the mechanical system under analysis.

Basing their works over the assumption that time series can be interpreted as images and the success obtained in computer vision tasks, approaches based on Convolutional Neural Networks (CNN) were explored for RUL estimation. Babu et al. [29] and Li et al. [30] explored deep CNN-based methods, whereas Cornelius et al. [31] leverages heteroscedastic and epistemic uncertainties to improve the RUL estimation in a deep CNN-based network. Nonetheless, the temporal dependencies occurring in sensor data hardly are learned by these techniques.

Therefore, sequence models (*e.g.* LSTM networks) are often exploited because of their ability to model the evolution of the measured features. Both a Multilayer Perceptron and a RNN were used in [32]. Due to the length of the time series considered in this field, RNNs can have problems remembering the important information and capturing long term dependencies, which has encouraged several researchers to exploit memory-based networks to store key information, such as GRUs [33, 34, 35] or LSTMs [13, 14, 15, 17, 33, 36]. Recently, [16] proposed an ensemble of bidirectional LSTM-based models, each trained on the input data framed with windows of a different size, in order to have each model focus on temporal dependencies that require more or less time to develop. A sequence model which was only recently used in some works [18, 19] on RUL estimation consists in the NTM. Graves et al. introduced this sequence model in [20], where they show that NTMs outper-

form LSTM networks in five tasks with increasing complexity, including: the "copy" task, which requires the model to observe a sequence and copy it in output; the "repeated copy", which is an extension of the previous task and asks the model to repeat the copy a given number of times; the "associative recall" task, which resembles sequence modeling and consists in recalling an input item that follows a given "query" item in a previously processed list of items; the "dynamic n-gram", which tests the capabilities of the models to learn a probability distribution; finally, the "priority sort" task, which requires the model to learn how to sort a sequence based on a priority vector. NTMs were introduced within the RUL estimation field by Falcon et al. in [18] and [19], where they obtained a lower prediction error than previously published works. Yet, in these works the NTM is used within bigger and complex architectures, so it is not clear how much the NTM is helping the whole approach. Conversely, in this paper the attention is driven towards the contribution given by the NTM when used as the whole feature extraction mechanism. In particular, even by using a single NTM as the feature extractor better performance are achieved than more popular LSTM-based methods, while also using around 28% fewer learnable parameters. Furthermore, these results are empirically validated by performing multiple experiments on two public datasets, the C-MAPSS dataset [6] and the PHM Society 2020 Data Challenge dataset [9].

3. The proposed Approach

A schematic overview of the proposed approach is shown in Fig. 1. First of all, the raw input time series are preprocessed following three simple steps: they are normalized using MinMax, i.e. $x'_i = \frac{2(x_i - X_{min})}{X_{max} - X_{min}} - 1$ where X_{max}, X_{min} represent maximum and minimum value of feature X; labeled, during training, with a piece-wise linear degradation function [32], limiting the RUL to 125; and cut into shorter time series using a sliding window technique. After the preprocessing, they are fed to the cells of the NTM. The hidden states computed by the NTM are interpreted as the feature vectors for the given time series. Each feature vector is then mapped to estimate RUL values through a simple decoder made of two stacked fully connected layers.

3.1. Neural Turing Machine

In the NTM, shown in Fig. 2, the memory bank $M \in \mathbb{R}^{l \times s}$ is made of l memory locations, i.e. $M = [m_1, m_2, ..., m_l]$, where each of them is a

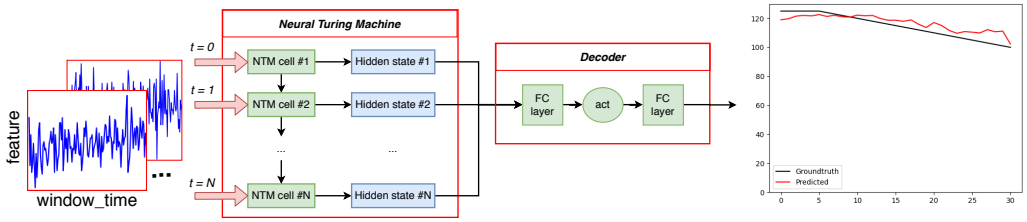

Figure 1: A graphical overview of the proposed approach. The time series are first cut into shorter windows, then fed to the network. The Neural Turing Machine is used as the feature extractor. Finally, two stacked fully connected networks are used to map the extracted features to a sequence of RUL values.

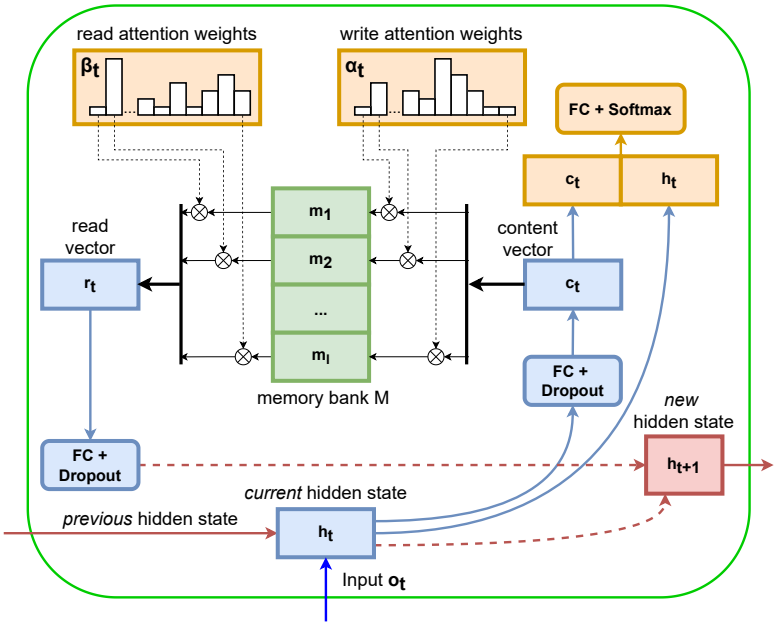

Figure 2: At time t, the NTM updates the hidden state h_{t+1} (red) by using its memory M, input vector o_t, and previous hidden state h_t. It is functionally separated into attention (yellow) with weights α_t and β_t, read/write operations (blue), and memory slots (green).

vector with s features, i.e. $m_i \in \mathbb{R}^{1\times s}$. Considering as input a windowed time series $T \in \mathbb{R}^{t_l \times f}$, made of t_l vectors of size f, the NTM sequentially processes T by extracting, storing, and eventually retrieving some of the most important information from each of the t_l measurement vectors using learnable read and write operations. In this way, the operations performed by the NTM use many memory vectors, whereas LSTM networks rely on one

memory vector and thus it is likelier that previously obtained information is rewritten and lost. The NTM also updates an hidden state $h_t \in \mathbb{R}^{1\times s}$ at each time step, which acts as a summary of the measurement vectors received so far and their interactions over time. The sequence of hidden states $H = \{h_i \,|\, i \in \mathbb{N}^+, i \leq t_l\}$ is then used as the automatically extracted features of T. Hence, the feature extraction process depends entirely on the NTM and its interactions with the raw sensor data, whereas Falcon et al. used the NTM on top of an LSTM [19] or in conjunction with a CNN and self-attention layers [18], therefore making unclear the contribution given by the NTM in the whole approach. As shown in Fig. 2 the input to the NTM at time step t is represented by $o_t \in \mathbb{R}^{1\times f}$ and consists of the measured value of f different sensors. Three types of operation are sequentially performed by the NTM: write, read, and hidden state update.

Write operation. At time t, the memory bank is updated by writing new information obtained by the current sensor measurements o_t and the previous hidden state h_{t-1}. In this way, it stores new knowledge from the input, while maintaining some of the previously captured information. The information to be written into the memory, i.e. the content vector $c_t \in \mathbb{R}^{1\times s}$, is computed as follows:

$$c_t = \delta_{p_1}(\sigma([o_t, h_{t-1}]W_{hc} + b_c)) \tag{1}$$

where δ_{p_1} is the dropout [37] operator with probability $p_1 \in [0, 1]$, $[\cdot]$ is the concatenation operator, $o_t \in \mathbb{R}^{1\times f}$ represents the current sensor measurements vector, and $h_{t-1} \in \mathbb{R}^{1\times s}$ the previous hidden state. $W_{hc} \in \mathbb{R}^{(f+s)\times s}$ and $b_c \in \mathbb{R}^{1\times s}$ are trainable parameters. To determine how much the memory should be modified, an attention mechanism is used in order to compute a weight for each of the l memory locations. This is done as following:

$$a_t = \delta_{p_1}(v_a \; tanh([c_t, h_{t-1}]W_{ha} + b_a)) \tag{2}$$

$$\alpha_{t,i} = \frac{exp(a_{t,i})}{\sum_{j=1}^{l} exp(a_{t,j})} \text{ for } i = 1, ..., l \tag{3}$$

In particular, $\alpha_t = \{\alpha_{t,1}, \ldots, \alpha_{t,i}, \ldots, \alpha_{t,l}\}$ represents the attention weights and Eq. 3 satisfies $\sum_i \alpha_{t,i} = 1$. $W_{ha} \in \mathbb{R}^{(s+s)\times l}$, $v_a \in \mathbb{R}$, and $b_a \in \mathbb{R}^{1\times l}$ are trainable parameters. Differently from previously published NTM-based networks, the dropout operator is added both in Eq. 1 and Eq. 2 in order

to mitigate overfitting situations and to improve the generalizability. Each memory slot m_i is then updated in the following way:

$$m_i = \alpha_{t,i} c_t + (1 - \alpha_{t,i}) m_i \text{ for } i = 1, ..., l \tag{4}$$

using the attention weights to balance how much of the new information (in c_t) should be saved inside the memory. This also helps the model to understand the underlying relations between the evolution of the different sensors: the new information is spread unevenly throughout all the memory locations, making it possible to store in each of them different key information.

Read Operation. After the update of the memory bank M, the NTM reads from M the new data which is used to update the hidden state. This operation is performed in a similar way to the write operation. First of all, as in Eq. 2 and Eq. 3, attention weights are computed on c_t and h_{t-1}. As for the write operation, the computation of the attention is regularized by the dropout operator. Then, the attention weights are used to compute a weighted average of the vectors currently contained in the memory, obtaining a read vector r_t. Since M contains information gathered from all the sensor measurements and their interaction over time, r_t represents an informative summary of the current health-related state of the mechanical system.

Dropout-augmented read and write operations. As mentioned before, both the write and read operations are augmented in this work by adding the dropout operator. As shown by Srivastava et al., this operator may mitigate overfitting and improve the performance on many heterogeneous tasks [37]. This is possible by reducing the co-adaptation: at training time, the parameters of the neurons are updated in a way such that they try to fix the mistakes made by other neurons, i.e. they co-adapt. By using the dropout, a fraction of the neurons is randomly shut off at training time, therefore making the neurons less prone to rely on co-adaptation. Since this phenomenon is unlikely to generalize, reducing it may lead to improved generalization.

Hidden State Update. Finally, the hidden state is updated with the knowledge gathered from the updated memory bank, the previous hidden state, and the current input measurements:

$$h_t = \sigma(o_t W_{oh} + r_t W_{rh} + h_{t-1} W_{hh} + b_h) \tag{5}$$

where $W_{oh} \in \mathbb{R}^{f \times s}$, $W_{rh} \in \mathbb{R}^{s \times s}$, $W_{hh} \in \mathbb{R}^{s \times s}$, and $b_h \in \mathbb{R}^s$ are trainable parameters. The sequence of hidden states $h_1, h_2, \ldots, h_N$ are grouped in a matrix $H \in \mathbb{R}^{t_l \times s}$ and used as the automatically extracted features for the input time series.

3.2. RUL Decoder

After the features are extracted by the NTM, a simple decoder (see Fig. 1) is employed to learn a mapping from these features to RUL values:

$$d_1 = \delta_{p_2}(\sigma(HW_{d_h} + b_{d_h})) \tag{6}$$

$$d_2 = d_1 W_{d_o} + b_{d_o} \tag{7}$$

where $W_{d_h} \in \mathbb{R}^{s \times fc}$, $W_{d_o} \in \mathbb{R}^{fc \times 1}$, $b_{d_h} \in \mathbb{R}^{fc}$, and $b_{d_o} \in \mathbb{R}$ are trainable parameters, and σ is the sigmoid function. As the output of this step, $d_2 \in \mathbb{R}^{t_l \times 1}$ is obtained, which represents the sequence of predicted RUL values.

3.3. Loss function

To train the model and obtain optimal weights and biases from a labelled and preprocessed training dataset, the Mean Square Error (MSE) of the predicted RUL is optimized with respect to the groundtruth values. It is defined as: $MSE = \frac{1}{n}\sum_{i=1}^{n}(RUL'_i - RUL_i)^2$, where n is the total number of data samples, RUL'_i and RUL_i represent respectively the predicted and groundtruth RUL for the i-th data point.

4. Experimental Results

4.1. Datasets under analysis

To evaluate the proposed methodology, two public datasets are considered: the NASA C-MAPSS Turbofan Engine Degradation Simulation Dataset [6] and the PHM Society 2020 Data Challenge [9].

The **C-MAPSS** dataset consists of 4 subdatasets (named FD001, FD002, FD003, and FD004) of multiple multivariate time series. Every measurement vector contained in each series is made of three operational settings and 21 sensor values, each recording distinct physical characteristics of the considered system. These include temperature and pressure measured at the fan inlet, and the temperature measured for each module in the gas path (HPC, HPT, and LPT). A full list of the considered characteristics can be found in Table 1 (from Saxena et al. [7]), whereas Figure 3 (left) presents a simplified diagram of the turbofan engine. The data points come from different turbofan engines of the same type, which have different levels of initial wear. While training series are run-to-failure, testing series have a positive RUL which represents the target label. Table 2 shows a summary of the number of time series, fault and operational conditions found in each subdataset.

Description of the characteristic	Units
Total temperature at fan inlet	°R
Total temperature at LPC outlet	°R
Total temperature at HPC outlet	°R
Total temperature at LPT outlet	°R
Pressure at fan inlet	psia
Total pressure in bypass-duct	psia
Total pressure at HPC outlet	psia
Physical fan speed	rpm
Physical core speed	rpm
Engine pressure ratio	–
Static pressure at HPC outlet	psia
Ratio of fuel flow to static pressure at HPC outlet	pps/psi
Corrected fan speed	rpm
Corrected core speed	rpm
Bypass ratio	–
Burner fuel-air ratio	–
Bleed enthalpy	–
Demanded fan speed	rpm
Demanded corrected fan speed	rpm
HPC coolant bleed	lbm/s
LPT coolant bleed	lbm/s

Table 1: Description of the 21 sensors available in the C-MAPSS dataset, from [7].

In the experiments performed in this paper, some of the raw input features are ignored because they are constant and thus not informative. In particular, 14 sensor measurements out of the total 21 sensors are kept, whose indices are 2-4, 7-9, 11-15, 17, and 20-21. A similar selection is also done in [15, 19, 38].

Moreover, the three operational settings can be used in datasets FD002 and FD004 to identify six operational conditions, as reported in [6]. These settings affect the measurements because an engine behaves differently based on its operational condition (*e.g.* whether the airplane is taking off or it is cruising above the clouds). Hence, for these two datasets, KMeans [39] is used to cluster the measurements into six groups. Then, the data within each cluster are normalized using MinMax, in order to use the same scale to treat data sampled in the same condition [29]. Each of the measurement vectors in FD002 and FD004 is then augmented concatenating the one-hot encoding of the operational condition, thus increasing the input size from 14

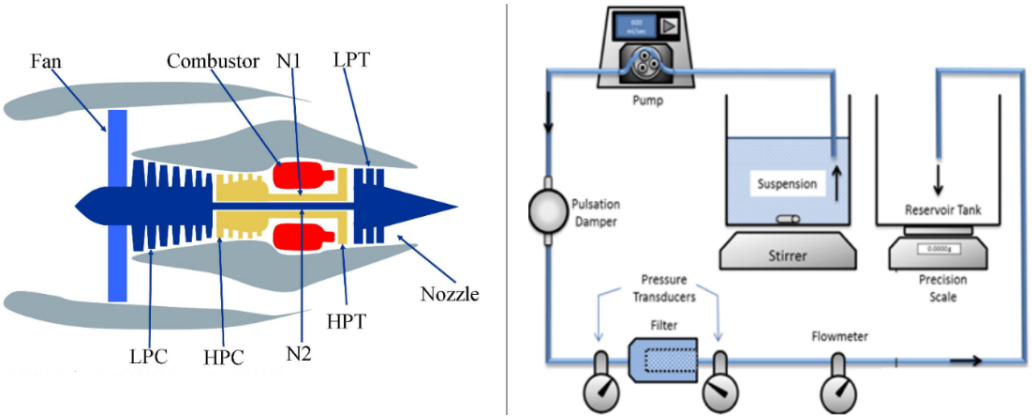

Figure 3: (left) Diagram of the turbofan engines considered for the C-MAPSS dataset [7]. (right) Diagram of the experimental rig used for the PHM Society 2020 challenge [9].

to 20. A similar data preprocessing approach was also reported in [13, 18, 29].

Finally, a value t_l for the window size is decided and used over all the four datasets. To determine t_l, five different values are tested: 30, 40, 50, 60, and 70. For easier reproducibility, Table 2 reports the amount of windowed time series observed during training after the train/validation split.

The second dataset, abbreviated to "**PHM20**", is released as part of the PHM Society 2020 Data Challenge [9] and consists of sensor measurements collected from an experimental rig used to simulate failures in a particle filtration system, which are widely used in industrial environments. This type of system is subject to clogging due to the presence of contaminants in the liquids and, in this case study, such a clog can be identified when the pressure difference is higher than 20 psi. The public dataset consists of 24 experiments for training and 8 for validation. Each experiment is annotated with the concentration (from 40% up to 47.5% with 2.5% increments) and the size of the particles (in the range 45-53um, or in the range 63-75um), and consists of several thousands of measurements, sampled at 10 Hz. Each sample is annotated with three sensors: the flow rate measured with a flowmeter, and both the upstream and downstream pressures which are measured with pressure transducers. A schematic of the experimental rig is shown in Fig. 3 (right). For each time step, five input features are considered: the three sensors, the concentration value, and the size of the particles. Then, these values are normalized with MinMax. In this work, the RUL is considered to be 0 when the pressure difference is higher than 20 psi and the time series are labelled

Dataset	**FD001**	**FD002**	**FD003**	**FD004**
Train time series	100	260	100	248
Test time series	100	259	100	248
Operating conditions	1	6	1	6
Fault conditions	1	1	2	2
Max length (testing)	303	367	475	486
Min length (testing)	31	21	38	19
Training samples with				
t_l=30	12100	32756	15499	38350
t_l=40	11400	30936	14799	36610
t_l=50	10700	29116	14099	34870
t_l=60	10000	27296	13399	33130
t_l=70	9300	25476	12699	31390

Table 2: Summary of the subdatasets of the C-MAPSS dataset.

with the piece-wise degradation function (with 125 as the maximum value), as in Ince et al. [40]. As in the previous case, five values for the window size t_l are considered but, since the time series in PHM20 are longer than those in C-MAPSS, these values are bigger than before: 70, 140, 210, 280, and 350.

4.2. Model evaluation

To evaluate the performance, the main metric used consists in the Root Mean Square Error (RMSE). Furthermore, for the C-MAPSS dataset the Scoring Function is also considered, whereas the Mean Absolute Error (MAE) is used for the PHM20 dataset.

4.2.1. Scoring Function

The Scoring Function was initially proposed in [6] and is defined as:

$$S = \sum_{i=1}^{n} s_i \text{ , where } s_i = \begin{cases} e^{\frac{-e_i}{13}} - 1, e_i < 0 \\ e^{\frac{e_i}{10}} - 1, e_i \geq 0 \end{cases} \tag{8}$$

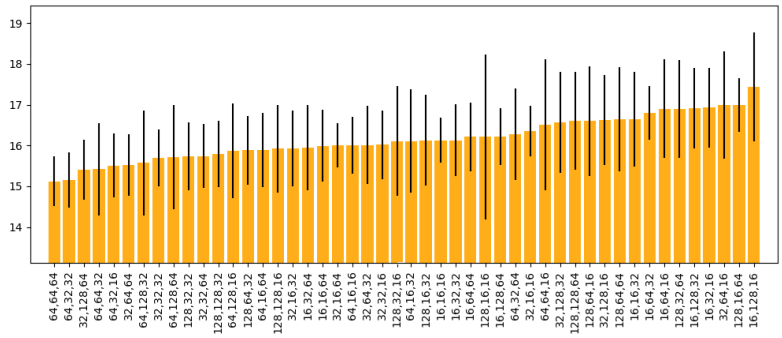

Figure 4: Average testing RMSE (with standard deviation) on FD001 using the best validation model. Hyperparameters s, l, and fc shown on x axis.

where S is the computed score, n is the total number of data samples, and $e_i = RUL'_i - RUL_i$ is the difference between estimated and groundtruth RUL at the i-th data point. The asymmetric nature of this function penalizes more the "too optimistic" predictions, meaning that it gives a higher score (i.e. worse) when the model predicts $\hat{v}$ but the true RUL is v such that $v < \hat{v}$, thus leading to an unpredicted early failure of the considered system. On the other hand, it penalizes less an "early" prediction (i.e. $\hat{v} < v$): although such prediction may trigger a superfluous maintenance, it should not lead to unexpected failures, possibly avoiding more severe consequences.

4.2.2. Root Mean Square Error and Mean Absolute Error

RMSE and MAE are commonly used to evaluate prediction accuracy, both giving equal weights for both early and late predictions. A key difference between the two metrics is how much they punish observations which are further from the mean: in particular, RMSE is a quadratic scoring rule, therefore it is more sensitive to large errors in the predictions.

4.3. Implementation details

The training is performed for 50 epochs, using a variable learning rate, starting from 0.005 and decaying it by a factor of 0.6 every 15 epochs. For C-MAPSS, the training time series (before cutting them into windows) are split with a 70/30 ratio to create the training and validation splits. For the PHM20 dataset, the original validation set is used as the testing set, and the training set is split with a 80/20 ratio to define the train and validation splits. 10 runs are performed and for each the best model on the validation set is selected

Method	s, l, fc values	Param. count	RMSE
LSTM [13]	32, 64, 8	31,761	16.14
Our NTM	32, 128, 64	22,945	$15.23_{\pm 0.66}$
LSTM	32, 48, 8	23,057	16.40
Our NTM	64, 64, 64	35,105	15.46

Table 3: Comparison of RMSE over FD001 with respect to the LSTM-based solution proposed in [13]. Better results, more efficiently (about 28% fewer parameters).

and used for testing. During training, the mini-batch gradient descent (batch size 100), and RMSProp (momentum 0.9, weight decay 0.0005) are employed. The dropout rates are set to $p_1 = 0.1$ and $p_2 = 0.25$.

The bias b_h is initialized to zero, and the weights v_a and v_b are sampled from a normal distribution with mean 0 and standard deviation 0.01. All the other weights and biases are initialized by sampling from a uniform distribution in the range $[-\sqrt{k}, \sqrt{k}]$, where $k = \frac{1}{in\_fts}$ and in_fts is the number of input features of the weight or bias (*e.g.* for $W_{hb} \in \mathbb{R}^{(s+s)\times l}$, $k = \frac{1}{s+s}$). The memory bank and hidden state are initialized with zeros.

Finally, PyTorch 1.3.0 is used to implement the proposed solution[1].

4.4. Discussion of the results on the CMAPSS dataset

Grid search results. To determine the best combination of the hyperparameters used in the proposed approach, i.e. the two sizes s and l of the NTM, and the hidden size, fc, of the decoder, multiple runs are performed on FD001. By using $fc = 8$, high RMSE values (around 12% higher than other combinations) are obtained, possibly implying that such a low number of neurons in the decoder is not enough to learn a meaningful RUL estimation function. Higher values for fc lead to better accuracy, although s and l influence the overall performance as well, as shown in Fig. 4.

Parameter efficiency of the NTM. By using the external memory, the NTM may learn better features with less parameters. To investigate this, a fair comparison to the LSTM-based solution proposed in [13] is made, because they perform hyperparameter optimization as well. Table 3 reports a lower estimation error for the NTM (15.23 compared to 16.14), while also using 28% fewer parameters (22945 compared to 31761), making it a better

[1]The code is released at: https://github.com/aranciokov/NTM-For-RULEstimation

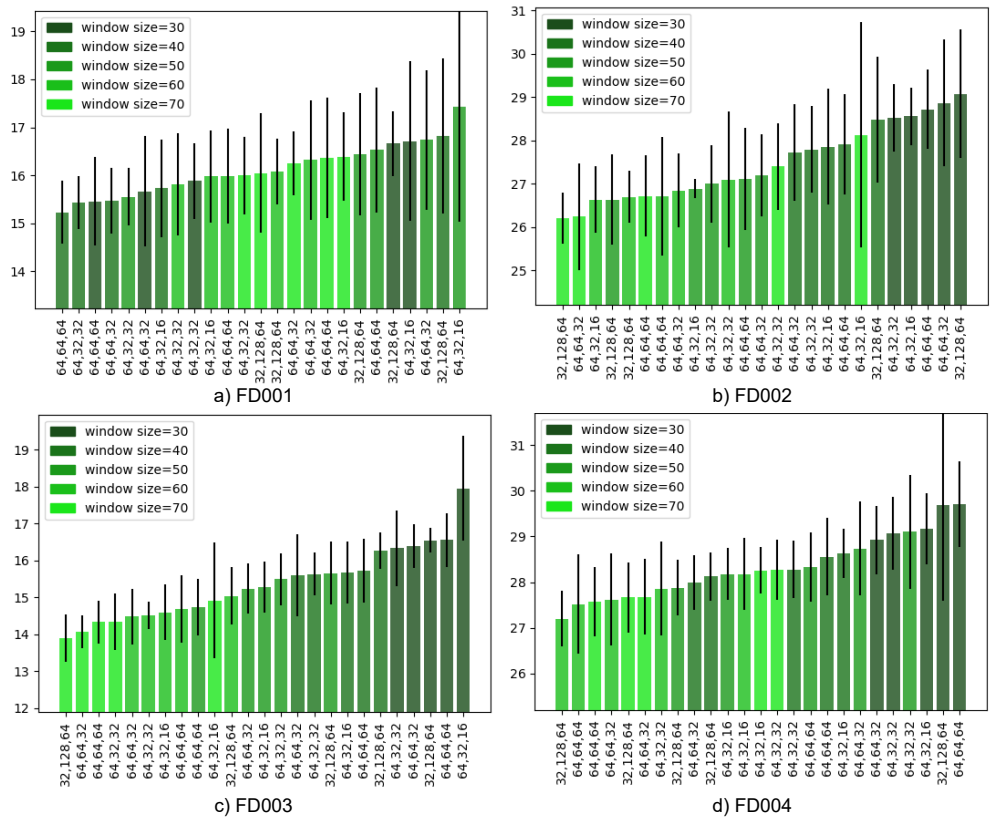

Figure 5: Average testing RMSE (with standard deviation) using five combinations of s, l, and fc (see Sec. 4.4), and five values for the window size. Over FD001, shorter time windows lead to better results, whereas longer windows are preferred for the other, more complex datasets. Best viewed in color.

approach when dealing with memory-constrained environments, *e.g.* embedded. Furthermore, even if the two networks had a similar amount of parameters, the NTM would still perform better, as reported in Table 3 (15.46 compared to 16.14 using around 32000 parameters, and 15.23 to 16.40 using 23000 parameters).

Window sizes. The size of the windows used during training (see Sec. 3) can be seen as another hyperparameter, since longer series may be more difficult to deal with but may also be more informative. Fig. 5 reports the error obtained over the four datasets using the five best combinations of hyperparameters (s, l, fc) found in the previous experiment, and five different

Methods	FD001	FD002	FD003	FD004
MLP [29]	1.8×10^4	7.8×10^6	1.7×10^4	5.6×10^6
CNN [29]	1.3×10^3	1.4×10^4	1.6×10^3	7.9×10^3
LSTM [13]	$\mathbf{3.4\times10^2}$	$\mathbf{4.4\times10^3}$	8.5×10^2	5.5×10^3
Our NTM	3.8×10^2	5.5×10^3	$\mathbf{3.0\times10^2}$	$\mathbf{5.2\times10^3}$
BiLSTM [14]	2.9×10^2	4.1×10^3	3.2×10^2	5.4×10^3
GADLM [38]	2.3×10^2	3.4×10^3	2.5×10^2	2.8×10^3
NTM-Hybrid [18]	2.1×10^2	6.0×10^3	2.7×10^2	4.8×10^3

Table 4: Comparison on the C-MAPSS dataset using the Scoring function (see Sec. 4.4 for the discussion).

sizes for the windows, i.e. 30, 40, 50, 60, and 70. For dataset FD001 (Fig. 5.a), short windows are more beneficial than longer time windows. The easier nature of this dataset may explain this, since the RUL values are influenced only by one operating condition and one fault condition. Conversely, for FD002 to FD004 these short time windows are not optimal, as the events which lead to a fault likely require more time to develop. All the following experiments use 70 as the window size.

Qualitative analysis. Figure 6 displays the predictions (blue) and groundtruth values (red) over the four datasets, showing that the proposed model can effectively estimate the RUL of unseen time series in the testing set. To give further evidence of this, Fig. 7 presents some examples of prediction on the testing set, showing that the RUL progression can be reliably predicted. Moreover, the accuracy increases as the fault gets closer.

Limitations of the piece-wise function. In the experiments presented both in this paper and in the literature, *e.g.* by Zheng et al. [13] and Babu et al. [29], the estimation error observed on FD002 and FD004 is higher than FD001 and FD003. This is mainly due to two factors. First, datasets FD002 and FD004 are more difficult, due to multiple operating conditions affecting the captured measurements. Secondly, the de facto standard degradation function [32] used to label the C-MAPSS dataset is not a perfect solution. In fact, since it limits the maximum RUL value observed during training, it is hard for any model to correctly predict at testing time RUL values which are higher than such a maximum. As an example, FD004 contains 67 (out of 248) times series with a RUL higher than 125 and the model fails these predictions (see Fig. 6.d). A new challenge is thus brought to light: the function proposed in [32] is in fact widely accepted and used [13, 14, 16, 18, 29, 30], yet this

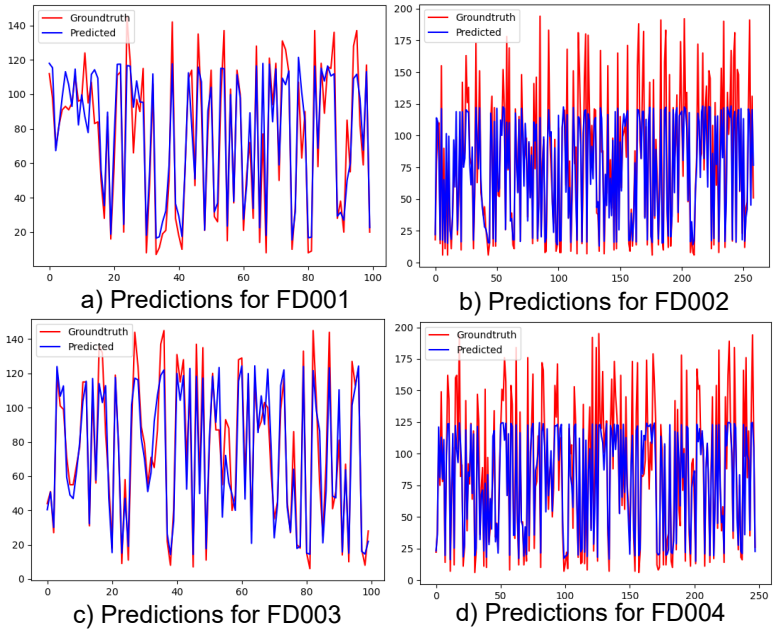

Figure 6: Comparison between predicted (blue) and groundtruth RUL values (red).

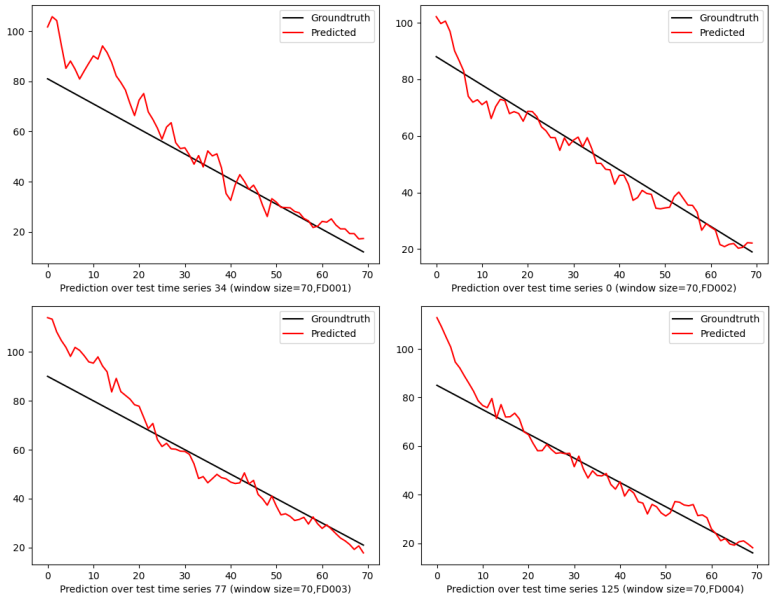

Figure 7: Examples of prediction (red) made by the proposed model during testing. The model generalizes well over unseen test examples. Note that the true RUL for testing series is only known for the last time step: a linear degradation is shown here for comparison.

Methods	FD001	FD002	FD003	FD004
MLP [29]	37.56	80.03	37.39	77.37
CNN [29]	18.45	30.29	19.82	29.16
LSTM [13]	16.14	**24.49**	16.18	28.17
Our NTM	$\mathbf{16.05_{\pm 1.2}}$	$26.21_{\pm 0.6}$	$\mathbf{13.90_{\pm 0.6}}$	$\mathbf{27.67_{\pm 0.7}}$
DCNN [30]	13.32	24.86	14.02	29.44
BiLSTM [14]	13.65	23.18	13.74	24.86
GADLM [38]	12.56	22.73	12.10	22.66
Sim-Sup [41]	18.33	-	12.73	-
MTW-BLSTM [16]	12.61	-	-	-
Multi-Local [42]	14.1	-	-	-
NTM-Hybrid [18]	12.53	27.04	13.73	28.11

Table 5: Comparison on the C-MAPSS dataset using the RMSE (see Sec. 4.4 for the discussion).

study shows it may not be the most appropriate solution for the task. Few works are actively working on alternative labeling functions, *e.g.* [15], but no definitive solutions are available to date.

Comparison with similar architectures. In Tables 4 and 5, the results obtained using $s = 32$, $l = 128$, $fc = 64$ are presented and compared to other published works. In particular, since the focus of this work is to explore the NTM as the main feature extraction component, it is more fair to compare to architectures following similar principles. As shown in both the Tables, models leveraging the sequential nature of the data, i.e. LSTM and NTM, obtain a lower estimation error. Moreover, the NTM excels in both metrics on the datasets involving multiple fault conditions (FD003 and FD004), and in terms of RMSE on FD001 (Table 5).

Comparison with more complex architectures. For a more comprehensive comparison, published works with more complex architectures, pretraining, *etc* are also considered in Tables 4 and 5. Compared to the deep CNN used by Li et al. [30], the NTM achieves a lower RMSE when multiple fault conditions affect the sensor measurements (13.90 and 27.67 compared to 14.02 and 29.44). Interestingly, although using only one direction to analyze the data, it achieves lower scores (3.0×10^2 and 5.2×10^3 compared to 3.2×10^2 and 5.4×10^3) than bidirectional LSTMs [14]. Multiple models (*e.g.* Xia et al. [16]) and additional pretraining (*e.g.* Ellefsen et al. [38]) lead to more accurate predictions, but these techniques could be also applied

s	l	fc	RMSE
16	128	64	11.35
32	128	64	7.03
64	128	64	5.87
128	128	64	7.10
64	16	64	10.11
64	32	64	5.50
64	**64**	64	5.45
64	128	64	5.87
64	64	16	16.46
64	64	32	7.32
64	64	**64**	5.45
64	64	128	6.12
64	**64**	**64**	**5.45**

Table 6: Hyperparameters optimization on PHM20 dataset.

to NTM-based solutions. Moreover, all the LSTM-based solutions could be further improved by replacing the LSTM with an NTM.

Finally, there are other recent papers employing LSTM (*e.g.* [17]) or CNN components (*e.g.* [43]) which perform better. Yet, a fair comparison is difficult to make: their experimental setting is different as the testing labels are also rectified by the piece-wise function, which is not done in this work.

4.5. Discussion of the results on the PHM20 dataset

Hyperparameters tuning. As for the C-MAPSS dataset, the influence of the hyperparameters of the NTM (s, l, and fc) is explored on the PHM20 dataset. Since it contains longer time series, the window size t_l is initially fixed to 210. Here, s, l, and fc are varied in {16, 32, 64, 128}. The average RMSE (on 10 runs) is reported in Table 6. Two observations can be made. First of all, 64 represents an optimal value among those analyzed for the three hyperparameters, leading to an RMSE of 5.45. Secondly, the size s of the memory locations and the number fc of neurons in the decoder are highly influential on the final performance.

Temporal context and external memory. By continuously interacting with the external memory, NTMs may be able to deal with longer time series than LSTMs. This may be strategic in industrial settings, where sensor measurements can be collected frequently over long periods of time.

	Our NTM		LSTM [13]	
window size t_l	MAE	RMSE	MAE	RMSE
70	4.44	6.82	4.97	7.05
140	4.54	6.74	5.38	7.77
210	3.74	5.45	5.09	7.30
280	**3.73**	**5.37**	5.45	7.59
350	4.97	6.93	5.45	8.42

Table 7: Comparison on PHM20 dataset between the proposed model and the LSTM-based model from [13]. Performance is measured both with RMSE and MAE.

To confirm this surmise, the performance of the two networks are compared as the length of the temporal context t_l increases. For the LSTM the same hyperparameters as in Zheng et al. [13] are used, whereas for the NTM $s = 64$, $l = 64$, and $fc = 64$. Table 7 reports the performance both in terms of MAE and RMSE. For the NTM, $t_l = 280$ represents an optimal value, whereas for the LSTM $t_l = 70$ leads to best results, although the error is far higher than the one achieved with the NTM. Two observations can be made. Firstly, the surmise is confirmed, since the NTM manages to deal with longer sequences, whereas the LSTM shows a decreasing accuracy as the sequences become longer. Secondly, the NTM has a better prediction capability than the LSTM on all the values tested for t_l, obtaining an estimation error as low as 5.37 RMSE and 3.73 MAE compared to 7.05 RMSE and 4.97 MAE obtained by the LSTM.

Comparison with state-of-the-art. Table 8 reports a comparison to state-of-the-art methods which took part into the PHM Society 2020 Data Challenge [9], alongside the LSTM-based model used in previous experiments. The winner of the challenge (Lomowski et al. [44]) used a non-comparable methodology, therefore it is not included here. Ince et al. [40] used Machine Learning techniques, including random forest and gradient boosting (implemented with Scikit-learn [45] and CatBoost [46]). For these two methods, the results obtained by running the public implementation provided by the authors are reported. With the proposed approach a lower estimation error is obtained, measured both with MAE and RMSE.

4.6. Discussion of NTM applicability to industrial contexts

The previous subsections show that a more accurate prediction is achieved if the NTM is used to automatically extract the features from the input

	MAE	RMSE
Random Forest	3.97	7.31
Gradient Boosting	3.82	6.81
LSTM	4.97	7.05
Our NTM	**3.73**	**5.37**

Table 8: Comparison with state-of-the-art methods on PHM20 dataset, including the first two methods from [40] and the results obtained by the LSTM-based model (based on [13]).

series. In particular, on both the C-MAPSS and the PHM20 dataset a lower estimation error is observed when compared to a popular LSTM-based model. Nonetheless, a superior accuracy may not be the only factor which needs to be taken into account when implementing a RUL estimation system.

Training times and size of the dataset. In a scenario in which the historical data form a sizable dataset, the NTM may require a bigger time investment to perform the training. As an example, on a system with a NVIDIA RTX A5000, an i7-9700K, and 32GB of RAM, training the NTM on the PHM20 dataset takes around 1 hour (single run), whereas it takes 10 minutes for the LSTM-based model. This is mainly due to the availability of a CUDNN implementation for the LSTM, which uses low level routines to reduce the running time of each operation. Conversely, the implementation of the NTM relies on higher level tools, making it slower. Nonetheless, the NTM and the LSTM share similar primitives, therefore a CUDNN implementation for the NTM is theoretically possible and may reduce the gap. With the current implementation, learning from very big datasets by means of a NTM-based system may become far too resource-consuming.

Low latency scenario. Industrial systems may need to predict the RUL with a negligible latency. In a similar scenario, the NTM may not be optimal. In fact, on the same system as before, the NTM estimates the RUL in around 55 ms (tested on 200 sequences of length 70), whereas the LSTM takes around 3 ms. As before, this is due to the usage of high level tools for the NTM, and it may be alleviated by changing implementation. Nonetheless, at the current state, if the RUL estimation system needs to perform a prediction with a really low delay (near real-time), then a LSTM-based implementation is preferable although with an inferior accuracy.

4.7. Summary of the main results

To ease the reading of the experimental section, the main results and major takeaways are summarized here:

- the NTM achieves lower estimation error than the LSTM, while also using fewer learnable parameters (hence, a smaller memory footprint);
- a longer temporal context is beneficial for the NTM, whereas the prediction accuracy of the LSTM worsens as the sequences become longer;
- the piece-wise degradation function commonly used to label the CMAPSS dataset limits the predictions made by the model, which becomes unable to predict high RUL values at testing time;
- the NTM performs better than approaches following a similar architecture, especially when multiple fault conditions affect the measurements;
- the simple architecture proposed in this paper competes with state-of-the-art approaches which use additional optimization steps and more complex or deeper architectures;
- it takes longer to train the NTM, when compared to the LSTM;
- the NTM provides a prediction with a longer delay than the LSTM.

5. Conclusions and future work

In this paper, the Neural Turing Machines are thoroughly analyzed for the Remaining Useful Life estimation problem. The advantages provided by having access to the additional memory are confirmed by an extensive experimental section which shows that the NTM can automatically extract useful features directly from the raw sensor measurements, obtaining better performance than MLP-, CNN-, and LSTM-based solutions: these performance are observed in terms of absolute estimation error, but also in three relative directions, that is parameter efficiency (fewer learnable parameters), memory efficiency (lesser memory footprint), and better usage of temporal context (longer sequences provide useful information to the NTM, whereas for the LSTM this does not hold). These results are empirically confirmed on two public datasets: the widely used C-MAPSS dataset [6] provided by NASA, and the recently released PHM Society 2020 Data Challenge [9]. The

evidences also suggest that the NTM outperforms the LSTM when multiple operating conditions and fault modes affect the sensor measurements: this may be strategic in industrial settings, since complex machinery contain several deteriorating components. Moreover, since these faults may require more time to develop, being able to learn from a longer temporal context may be fundamental to catch them before they happen, therefore raising further interest towards the NTM. Furthermore, even though in this work the NTM is used as the only feature extraction component, it can still achieve comparable and even competitive results to state-of-the-art techniques using ensemble of models, additional pretraining, *etc.* Therefore, combining the NTM with other state-of-the-art techniques may lead to additional improvements.

There is still room for research. In the experimental results, it is shown that the labeling function [32] commonly used for this task may not be the best choice and can highly affect the final performance. This is especially crucial for the datasets where the time series in the testing set have a higher groundtruth RUL than the maximum value used during the labeling step and therefore used to perform the training process. To address this, the community is actively seeking new solutions (*e.g.* Elsheikh et al. [15]) but, to date, no definitive solutions are found. Finally, some limitations of the NTM were also highlighted and contextualized to industrial scenarios, leaving further space for extensions and future works.

Acknowledgement

We thank the NASA Ames Prognostics Center of Excellence for the public release of the Turbofan Engine Degradation Simulation Data Set.

References

[1] A. K. Jardine, D. Lin, D. Banjevic, A review on machinery diagnostics and prognostics implementing condition-based maintenance, MSSP 20 (7) (2006) 1483–1510.

[2] B. M. Imam, M. K. Chryssanthopoulos, Causes and consequences of metallic bridge failures, Structural engineering international 22 (1) (2012) 93–98.

[3] P. Milillo, G. Giardina, D. Perissin, G. Milillo, A. Coletta, C. Terranova, Pre-collapse space geodetic observations of critical infrastructure: The morandi bridge, genoa, italy, Remote Sensing 11 (2019) 1403.

[4] W. Teng, C. Han, Y. Hu, X. Cheng, L. Song, Y. Liu, A robust model-based approach for bearing remaining useful life prognosis in wind turbines, IEEE Access 8 (2020) 47133–47143.

[5] N. Li, Y. Lei, T. Yan, N. Li, T. Han, A wiener-process-model-based method for remaining useful life prediction considering unit-to-unit variability, IEEE Transactions on Industrial Electronics 66 (3) (2018) 2092–2101.

[6] A. Saxena, K. Goebel, Turbofan engine degradation simulation data set, NASA Ames Prognostics Data Repository (2008).
URL `http://ti.arc.nasa.gov/project/prognostic-data-repository`

[7] A. Saxena, K. Goebel, D. Simon, N. Eklund, Damage propagation modeling for aircraft engine run-to-failure simulation, in: 2008 international conference on prognostics and health management, IEEE, 2008, pp. 1–9.

[8] A. Agogino, K. Goebel, Milling data set, available at http://ti.arc.nasa.gov/project/prognostic-data-repository (2007).

[9] P. Society, Phm society 2020 data challenge, https://phm-europe.org/data-challenge-2020 (last accessed: 23 April 2022) (2020).

[10] Y. Pan, R. Hong, J. Chen, W. Wu, A hybrid dbn-som-pf-based prognostic approach of remaining useful life for wind turbine gearbox, Renewable Energy 152 (2020) 138–154.

[11] Z. Chen, Y. Li, T. Xia, E. Pan, Hidden markov model with auto-correlated observations for remaining useful life prediction and optimal maintenance policy, Reliability Engineering & System Safety 184 (2019) 123–136.

[12] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[13] S. Zheng, K. Ristovski, A. Farahat, C. Gupta, Long short-term memory network for remaining useful life estimation, in: ICPHM, IEEE, 2017, pp. 88–95.

[14] J. Wang, G. Wen, S. Yang, Y. Liu, Remaining useful life estimation in prognostics using deep bidirectional lstm neural network, in: PHM-Chongqing, IEEE, 2018, pp. 1037–1042.

[15] A. Elsheikh, S. Yacout, M.-S. Ouali, Bidirectional handshaking lstm for remaining useful life prediction, Neurocomputing 323 (2019) 148–156.

[16] T. Xia, Y. Song, Y. Zheng, E. Pan, L. Xi, An ensemble framework based on convolutional bi-directional lstm with multiple time windows for remaining useful life estimation, Computers in Industry 115 (2020) 103182.

[17] J. Xia, Y. Feng, C. Lu, C. Fei, X. Xue, Lstm-based multi-layer self-attention method for remaining useful life estimation of mechanical systems, Engineering Failure Analysis 125 (2021) 105385.

[18] A. Falcon, G. D'Agostino, G. Serra, G. Brajnik, C. Tasso, A dual-stream architecture based on neural turing machine and attention for the remaining useful life estimation problem, in: PHM Society European Conference, Vol. 5, 2020, pp. 10–10.

[19] A. Falcon, G. D'Agostino, G. Serra, G. Brajnik, C. Tasso, A neural turing machine-based approach to remaining useful life estimation, in: ICPHM, 2020, pp. 1–8.

[20] A. Graves, G. Wayne, I. Danihelka, Neural turing machines, arXiv preprint arXiv:1410.5401 (2014).

[21] C. S. Byington, S. E. George, G. W. Nickerson, Prognostic issues for rotorcraft health and usage monitoring systems, A Critical Link: Diagnosis to Prognosis (1997) 93.

[22] F. L. Greitzer, L. J. Kangas, K. M. Terrones, M. A. Maynard, B. W. Wilson, R. A. Pawlowski, D. R. Sisk, N. B. Brown, Gas turbine engine health monitoring and prognostics, in: International Society of Logistics (SOLE) 1999 Symposium, Las Vegas, Nevada, Vol. 30, 1999, pp. 1–7.

[23] N. Bolander, H. Qiu, N. Eklund, E. Hindle, T. Rosenfeld, Physics-based remaining useful life prediction for aircraft engine bearing prognosis, in: Annual Conference of the PHM Society, Vol. 1, 2009.

[24] A. Coppe, M. J. Pais, R. T. Haftka, N. H. Kim, Using a simple crack growth model in predicting remaining useful life, Journal of Aircraft 49 (6) (2012) 1965–1973.

[25] C. Ordóñez, F. S. Lasheras, J. Roca-Pardiñas, F. J. de Cos Juez, A hybrid arima–svm model for the study of the remaining useful life of aircraft engines, Journal of Computational and Applied Mathematics 346 (2019) 184–191.

[26] Y. Zhou, M. Huang, Lithium-ion batteries remaining useful life prediction based on a mixture of empirical mode decomposition and arima model, Microelectronics Reliability 65 (2016) 265–273.

[27] K. Medjaher, D. A. Tobon-Mejia, N. Zerhouni, Remaining useful life estimation of critical components with application to bearings, IEEE Transactions on Reliability 61 (2) (2012) 292–302.

[28] H.-Z. Huang, H.-K. Wang, Y.-F. Li, L. Zhang, Z. Liu, Support vector machine based estimation of remaining useful life: current research status and future trends, Journal of Mechanical Science and Technology 29 (1) (2015) 151–163.

[29] G. S. Babu, P. Zhao, X.-L. Li, Deep convolutional neural network based regression approach for estimation of remaining useful life, in: DASFAA, Springer, 2016, pp. 214–228.

[30] X. Li, Q. Ding, J.-Q. Sun, Remaining useful life estimation in prognostics using deep convolution neural networks, Reliab. Eng. Syst. Saf. 172 (2018) 1–11.

[31] J. Cornelius, B. Brockner, S. H. Hong, Y. Wang, K. Pant, J. Ball, Estimating and leveraging uncertainties in deep learning for remaining useful life prediction in mechanical systems, in: ICPHM, 2020, pp. 1–8.

[32] F. O. Heimes, Recurrent neural networks for remaining useful life estimation., in: ICPHM, 2008.

[33] M. Baptista, H. Prendinger, E. Henriques, Prognostics in aeronautics with deep recurrent neural networks, in: PHM Society European Conference, Vol. 5, 2020.

[34] J. Wang, J. Yan, C. Li, R. X. Gao, R. Zhao, Deep heterogeneous gru model for predictive analytics in smart manufacturing: Application to tool wear prediction, Computers in Industry 111 (2019) 1–14.

[35] Q. Luo, Y. Chang, J. Chen, H. Jing, H. Lv, T. Pan, Multiple degradation mode analysis via gated recurrent unit mode recognizer and life predictors for complex equipment, Computers in Industry 123 (2020) 103332.

[36] B. Zhang, S. Zhang, W. Li, Bearing performance degradation assessment using long short-term memory recurrent network, Computers in Industry 106 (2019) 14–29.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, JMLR 15 (1) (2014) 1929–1958.

[38] A. L. Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, H. Zhang, Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture, Reliab. Eng. Syst. Saf. 183 (2019) 240–251.

[39] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, Vol. 1, Oakland, CA, USA, 1967.

[40] K. Ince, E. Sirkeci, Y. Genç, Remaining useful life prediction for experimental filtration system: A data challenge, in: PHM Society European Conference, Vol. 5, 2020.

[41] M. Hou, D. Pi, B. Li, Similarity-based deep learning approach for remaining useful life prediction, Measurement 159 (2020) 107788.

[42] J. Lyu, R. Ying, N. Lu, B. Zhang, Remaining useful life estimation with multiple local similarities, Eng. Appl. Artif. Intell. 95 (2020) 103849.

[43] L. Liu, L. Wang, Z. Yu, Remaining useful life estimation of aircraft engines based on deep convolution neural network and lightgbm combination model, International Journal of Computational Intelligence Systems 14 (1) (2021) 1–10.

[44] R. Łomowski, S. Hummel, A method to estimate the remaining useful life of a filter using a hybrid approach based on kernel regression and simple statistics, in: PHM Society European Conference, Vol. 5, 2020.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, the Journal of machine Learning research 12 (2011) 2825–2830.

[46] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, A. Gulin, Catboost: Unbiased boosting with categorical features. arxiv 2017, arXiv preprint arXiv:1706.09516 (2017).