A Neural Turing Machine-based approach to Remaining Useful Life Estimation

Alex Falcon*†, Giovanni D'Agostino†, Giuseppe Serra†, Giorgio Brajnik†, Carlo Tasso†
*Fondazione Bruno Kessler, Trento, Italy
†Artificial Intelligence Laboratory
Università degli Studi di Udine, Italy
Email: {falcon.alex, dagostino.giovanni}@spes.uniud.it
{giuseppe.serra, giorgio.brajnik, carlo.tasso}@uniud.it

***Abstract*—Estimating the Remaining Useful Life of a mechanical device is one of the most important problems in the Prognostics and Health Management field. Being able to reliably estimate such value can lead to an improvement of the maintenance scheduling and a reduction of the costs associated with it. Given the availability of high quality sensors able to measure several aspects of the components, it is possible to gather a huge amount of data which can be used to tune precise data-driven models. Deep learning approaches, especially those based on Long-Short Term Memory networks, achieved great results recently and thus seem to be capable of effectively dealing with the problem. A recent advancement in neural network architectures, which yielded noticeable improvements in several different fields, consists in the usage of an external memory which allows the model to store inferred fragments of knowledge that can be later accessed and manipulated. To further improve the precision obtained thus far, in this paper we propose a novel way to address the Remaining Useful Life estimation problem by giving an LSTM-based model the ability to interact with a content-based memory addressing system. To demonstrate the improvements obtainable by this model, we successfully used it to estimate the remaining useful life of a turbofan engine using a benchmark dataset published by NASA. Finally, we present an exhaustive comparison to several approaches in the literature.**

I. Introduction

The estimation of the Remaining Useful Life (RUL) of a mechanical device is one of the key problems studied in the field of Prognostics and Health Management (PHM) [1]. In fact, knowing in advance if and when a machine is going to have a failure is fundamental to plan in advance its maintenance, allowing the users of such machine to benefit the most from its working lifetime and limiting the losses caused by the unexpected failure of the machine and the need of repairing or replacing it [2, 3]. The typical methods used to estimate the RUL of mechanical devices belong to three categories: physics-based approaches (also called model-based approaches), data-driven approaches, and hybrid approaches [4].

Physics-based approaches involve the creation of a model that simulates the physical behaviour of the piece of equipment analyzed in order to predict how and when it will have a failure. Even though physics-based approaches do not require access to a large amount of data in order to estimate the RUL of a mechanical device, the difficulty, the time, and the costs involved in developing an accurate physical model to simulate the behaviour of the analyzed device, make the model-based approaches unfeasible when such equipment is complex [1, 5].

Data-driven approaches on the other hand typically use pattern recognition and machine learning to make an estimation of the RUL of the mechanical device that is being analyzed, thus requiring a large amount of data used to train a mathematical model from which the RUL can be predicted [6, 2]. In the last years, thanks to the advancements made in the field of deep learning, data-driven approaches based on the usage of neural networks have experienced an increasing popularity. Deep learning models based on Convolutional Neural Networks (CNN), Long Short-Term Memory networks (LSTM), and Recurrent Neural Networks (RNN) have in fact been used in the latest years to estimate the remaining useful life [7, 1, 8].

What are the previous approaches missing or what are they doing "wrong"? For RUL estimation problems, CNN-based approaches do not seem to be an appropriate choice, because they do not exploit the sequential nature of the data. RNN-based approaches on the other hand do exploit such nature, yet they have troubles dealing with the really long time series which need to be dealt with due to vanishing gradients. LSTM-based approaches seem to fit better, thanks to their inner capability of dealing with long sequences; yet, they are heavily based on hidden states, and they rely on the fact that such a hidden state can support learning of the most informative characteristics of the data. We thought of boosting the mnemonic capabilities of the LSTM networks through the use of an external memory support, much like in a personal computer the utilization of a RAM boosts the capabilities of a CPU (otherwise limited by its own restricted caching storage). The intuition behind such idea is that enriching LSTM networks with an external memory support, where to store inferred fragments of knowledge, can improve the ability of the system to focus on relevant characteristics of the data. To test such a hypothesis, we decided to tackle the problem of RUL estimation through a novel deep learning approach based

978-1-7281-6286-7/20/$31.00 ©2020 IEEE

on the exploitation of a Neural Turing Machine combined with LSTM networks.

Neural Turing Machines (NTM) were introduced by Graves *et al.* as a working memory system designed to solve tasks that require the application of approximate rules to data that are quickly bound to memory slots (the so-called "rapidly-created variables") [9]. NTMs can be seen as a way to give neural networks the ability to interact with a memory, similarly to what happens in traditional computers with the random-access memory. Since their introduction, NTMs have been used with success in research fields such as Video Question Answering (VQA) [10] and for the sequential learning of human mobility patterns [11]. We claim that giving the model the ability to learn what pieces of knowledge to store in an external memory (the NTM) and when to retrieve them, improves the model accuracy in estimating the RUL of a given machine at a given time.

Our major contributions can be summarized as follows:

- We propose a novel neural network architecture for the RUL estimation based on the usage of an external memory module. Our hypothesis is that the model can profit from using another memory where processed hidden features can be selectively stored, manipulated, and retrieved. To evaluate our hypothesis, we present an exhaustive comparison with several architectures available in the literature.
- We show that our model achieves a low estimation error with respect to the other single-stream architectures, while still being comparable to recent dual-stream architectures.

The rest of the paper is organized as follows: Section II presents the RUL estimation problem setting and the tools used in our approach. Section III describes our approach, while in Section IV are illustrated the results of our experiments on a benchmark dataset. Section V concludes the paper with a discussion of the results and future work.

II. Related Work

A well known benchmark dataset for the RUL estimation problem is the C-MAPSS dataset [12], which is a dataset published by NASA that consists of several time series of sensor measurements of different turbofan engines. A first machine learning approach to the problem was given by Heimes in [13], where the author used a Multilayer Perceptron and a RNN.

In [7], Babu *et al.* proposed a deep learning approach consisting in two CNN layers alternated with pooling layers, followed by a Feedforward network used to estimate the RUL value. In a more recent work, [8] follows a similar approach using a deeper network with more CNN layers.

Given the sequential nature of the data, an architecture based on LSTM networks was proposed by Zheng *et al.* in [1]. To analyze the time series in both directions, [14] and [15] proposed to use a Bidirectional LSTM-based network, achieving further improvements with respect to the previous works. Zhang *et al.* in [16] proposed to use a Deep Belief Networks (DBN) ensemble method with two conflicting objectives (accuracy and diversity), where evolutionary algorithms are integrated with the traditional training algorithms to first train multiple DBNs and then combine them. Ellefsen *et al.* in [17] proposed to use an architecture composed of Restricted Boltzmann Machines, LSTM, and Feedforward networks, while following a semi-supervised training technique. Moreover, the authors used genetic algorithms to select the best hyper-parameters (such as the activation functions in the network, and the number of neurons in the hidden layers), showing that it is a valid approach to effectively tune such hyper-parameters.

Whereas the above mentioned papers use either an LSTM-based or a CNN-based network, another type of approach involves the use of multiple streams. Double stream approaches were tested on the C-MAPSS dataset by Li *et al.* in [4], and by Al-Dulaimi *et al.* in [18]. In these architectures, the available time series are usually cut time-wise in order to obtain shorter, fixed-size windows, which are later fed to both an LSTM-based network on one stream, and a CNN-based network on the other stream. In particular, the network used in [4] adopts a single LSTM on one stream, a CNN followed by a pooling and a flatten layer on the other stream, then sums element-wise the feature vectors, feeding the result to another LSTM followed by a Feedforward layer. On the other hand, the network used in [18] adopts three stacked LSTM layers on one stream, two stacked CNN plus pooling layers followed by a CNN plus flattening layer on the other stream, followed by three Feedforward layers which act as a fusion layer. Although the double-stream approaches obtain good results, the magnitude of the improvements introduced with respect to the network complexity of such architectures are not large enough to allow us to ignore the validity of single-stream architectures.

III. Our Approach

A. Overview of the approach

A graphical overview of the proposed approach can be seen in Figure 1. First of all, as described in the previous subsections, the raw input time series are preprocessed thus obtaining a set of shorter windowed time series. Each of these windows is then given in input to the two stacked LSTM networks, obtaining a sequence of extracted features. These features are then concatenated by the NTM to a new set of updated features computed by the NTM itself. The resulting object is then mapped to estimated RUL values by two stacked Feedforward layers.

B. The Proposed Model

The data typically considered in the Prognostics and Health Management field is composed of long time series measurements of sensors data. To model temporally-related sequential data and the evolution of its intrinsic characteristics, Recurrent Neural Networks have shown good performance in extrapolating hidden patterns in data. Due to the wide extent of the sequences considered in the RUL estimation problem, the issues of exploding and vanishing gradients need to be paid attention to [19]. In particular, to deal with such issues a

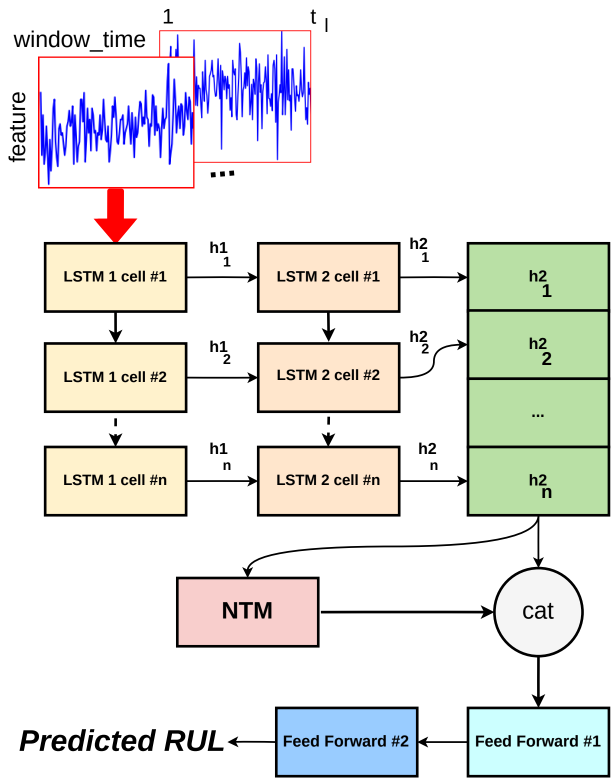

Fig. 1. A graphical overview of our approach. The time series are first cut into shorter windows, then given as input to the network. Following the two stacked LSTM networks, the extracted features are concatenated to the augmented features computed by the NTM module. At the end, two stacked Feedforward networks are used to map the extracted features to a sequence of RUL values.

type of RNN called Long Short-Term Memory Networks [20] was introduced. In this particular type of RNN, the flow of information inside and between the cells of an LSTM network at time t is controlled by three gates: the input gate i_t, the output gate o_t, and the forget gate f_t. The input gate decides whether to update the state of the LSTM by using the current input x_t, the forget gate decides whether to keep or forget the information represented in the previous state of the LSTM c_{t-1}, and the output gate decides whether to pass or not the updated hidden state h_t to the next cell of the network. The new state of the LSTM c_t is calculated by summing the previous state of the LSTM c_{t-1} with the new gated input (see Eq.4). The interactions between the gates, the cell states, and the hidden states of the LSTM networks are illustrated in Fig. 2. The LSTM equations are the following:

$$i_t = \sigma(W_i x_t + H_i h_{t-1} + b_i) \tag{1}$$

$$o_t = \sigma(W_o x_t + H_o h_{t-1} + b_o) \tag{2}$$

$$f_t = \sigma(W_f x_t + H_f h_{t-1} + b_f) \tag{3}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ tanh(W_c x_t + H_c h_{t-1} + b_c) \tag{4}$$

$$h_t = o_t \circ tanh(c_t) \tag{5}$$

where W_*, H_*, and b_* are respectively the trainable weights and biases for the input, output, forget gates and for updating

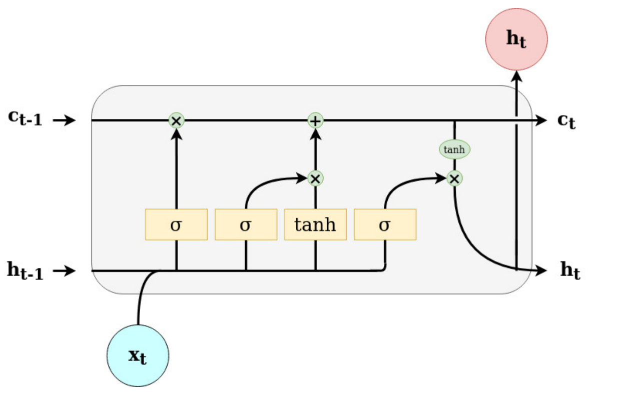

Fig. 2. The flow of information inside an LSTM cell at time t. σ denotes the sigmoid function, x_t the input vector, c_{t-1} the previous cell state and h_{t-1} the previous hidden state.

the cell state. h_{t-1} and h_t are respectively the hidden layer activation of the previous and of the current iteration, while c_{t-1} and c_t are respectively the cell states of the previous and of the current cell of the network. x_t denotes the current input, while f_t, o_t and i_t are the gate activations. $\circ$ represents the element-wise multiplication operator (Hadamard product). Finally, σ represents the sigmoid activation function.

Given the sequential nature and the considerable length of the time series data at hand, we opted for an LSTM-based network. Moreover, we considered the Feedforward networks to be the best solution to learn a mapping from the sequence of hidden features extracted by the LSTM network to RUL values. In particular, we decided to use as a starting point a network composed of two stacked LSTM networks followed by two stacked Feedforward layers, as this was shown to be a suitable solution in previous works [1] and it was shown to have better performance than a network composed of a single LSTM followed by a single Feedforward network [21].

To further boost the memory capabilities of our network, we are coupling the LSTM network with an NTM. We claim that the availability of an external memory where the processed hidden features can be selectively stored, manipulated, and retrieved can help the model to better understand the hidden patterns in the data and thus improve the capabilities of the subsequent direct RUL mapping module. The external memory used in our approach is based on the one implemented by Fan *et al.* in [10], which was customized to fit its architecture to the RUL estimation problem. The Neural Turing Machine is constituted by a memory module and a controller unit, where the memory module is made up of memory slots $M = [m_1, m_2, ..., m_S]$ and a memory hidden state h^m, while the controller unit consists of a Feedforward network; the inputs to the memory are represented by the vector o^m. The external memory supports three types of operations: write operations, read operations, and hidden state updates.

1) Write Operation: The content to be written into the memory at time t is represented by the content vector c_t^m and is computed as follows:

$$c_t^m = \sigma(W_{oc}^m o_t^m + W_{hc}^m h_{t-1}^m + b_c^m) \tag{6}$$

where o_t^m represents the current input vector, h_{t-1}^m the previous hidden state. W_{oc}^m and W_{hc}^m represent the trainable weights, and b_c^m represents the bias. The weights to be written into the memory slots of the NTM are defined as $\alpha_t = \{\alpha_{t,1}...\alpha_{t,i}...\alpha_{t,S}\}$ such that:

$$a_t^m = v_a^\top tanh(W_{ca}^m c_t^m + W_{ha}^m h_{t-1}^m + b_a^m) \tag{7}$$

and

$$\alpha_{t,i} = \frac{exp(a_{t,i})}{\sum_{j=1}^{S} exp(a_{t,j})} \text{ for } i = 1, ..., S \tag{8}$$

satisfying $\sum_i \alpha_{t,i} = 1$. W_{ca}^m, W_{ha}^m, and $v_a^\top$ represent the trainable weights, and b_a^m represents the bias. Each memory slot m_i is then updated in the following way:

$$m_i = \alpha_{t,i} c_t^m + (1 - \alpha_{t,i}) m_i \text{ for } i = 1, ..., S \tag{9}$$

2) Read Operation: The next step for the memory module is to read from the memory slots M. The normalized attention weights $\beta_t = \{\beta_{t,1}...\beta_{t,i}...\beta_{t,S}\}$ are such that:

$$b_t^m = v_b^\top tanh(W_{cb}^m c_t^m + W_{hb}^m h_{t-1}^m + b_b^m) \tag{10}$$

and

$$\beta_{t,i} = \frac{exp(b_{t,i})}{\sum_{j=1}^{S} exp(b_{t,j})} \text{ for } i = 1, ..., S \tag{11}$$

where W_{cb}^m, W_{hb}^m, and $v_b^\top$ represent the trainable weights, and b_b^m represents the bias. The content r_t read from the external memory is the weighted sum of each memory slot content:

$$r_t = \sum_{i=1}^{S} \beta_{t,i} \cdot m_i \tag{12}$$

3) Hidden State Update: After performing the write and read operations, the final task of the external memory at the t-th iteration is to update its hidden state h_t^m as following:

$$h_t^m = \sigma(W_{oh}^m o_t^m + W_{rh}^m r_t + W_{hh}^m h_{t-1}^m + b_h) \tag{13}$$

where W_{oh}^m, W_{rh}^m, and W_{hh}^m represent trainable weights, and b_a^m represents the bias.

C. Data preprocessing

The data used in the experiment proposed in this paper come from the FD001 subdataset of the C-MAPSS dataset. Such data has been preprocessed by following three steps: by normalizing the data with a z-score normalization, by defining a target function for the RUL and by using it to label the data, and by cutting the time series using a sliding time window approach.

1) Z-Score Normalization: Time series of sensor measurements usually range between multiple scales. A normalization step was hence performed to convert all these features into a common scale. In particular, we are using the z-score normalization, described as:

$$Norm(x) = \frac{x - \mu}{\sigma} \tag{14}$$

where μ and σ represent respectively the mean value and the standard deviation of the feature x. By doing so, the data points are such that their mean is zero, whereas their standard deviation equals to one.

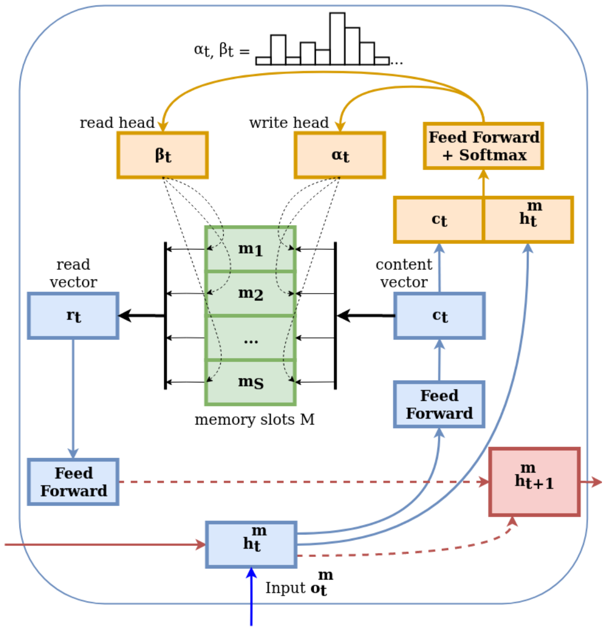

Fig. 3. The external memory of our architecture at time t with memory slots $M = [m_1, m_2, ..., m_S]$, read and write heads α_t and β_t, input vector o_t^m, and hidden state h_t^m.

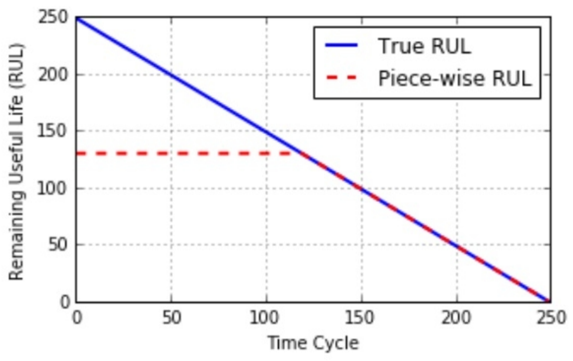

Fig. 4. The piece-wise linear RUL target function [1].

2) RUL Target Function: We apply a piece-wise linear RUL target function (pictured in Fig. 4) to represent the remaining useful life of an engine, which limits the maximum value of the RUL function to 130. This limitation is made in order to prevent the learning algorithm from overestimating the RUL; the piece-wise linear function is often regarded as the most logical model to represent the degradation of an engine, as the degradation of the analyzed system typically starts only after a certain degree of usage [13, 7].

3) Sliding time window: The sliding time window approach is used to process the run-to-failure data to give as an input to the learning model. This preprocessing step was done in order to extract data so that the input of the prediction model has a fixed length. As shown in Fig. 1, the signal has n features and the signal length is L. The data is extracted by sliding a time window of size t_l, and the sliding step size equals to one. The size of the array extracted each time by the time window

TABLE I
SUMMARY OF THE FD001 SUBDATASET OF THE C-MAPSS DATASET.

Dataset	FD001
Train trajectories	100
Test trajectories	100
Operating conditions	1
Fault conditions	1

is $t_l \times n$ (length of the time window $\times$ numbers of features), the total number of arrays is $L - t_l$ (life span - time window length), and the output for each window is the corresponding series of RUL values. In particular, we set the window length $t_l = 30$ in line with other works, such as [18] and [4].

D. Loss function

To train the model and obtain optimal weights and biases, we are considering the Mean Square Error (MSE) as the loss function to minimize. It is defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(RUL'_i - RUL_i)^2 \tag{15}$$

where MSE is the computed MSE value, N is the total number of testing data samples, RUL'_i and RUL_i represent respectively the estimated RUL and the groundtruth RUL, with respect to the i-th data point.

IV. Experimental Results

A. The C-MAPSS Dataset

For our experiment and problem setting, we considered the well-known NASA C-MAPSS (Commercial Modular Aero-Propulsion System Simulations) Turbofan Engine Degradation Simulation Dataset [12]. This dataset includes 4 subdatasets called FD001, FD002, FD003 and FD004, consisting of multiple multivariate time series. The data of such time series come from the sensors of different engines of the same type. Considering that this work is a first approach in the utilization of NTMs in the field of RUL estimation, at the moment we have focused on the first subdataset (FD001); current work in progress deals with the other datasets. A summary about the number of time series (called *trajectories*), fault conditions and operational conditions in the datasets is available in Table I. During our experiments, we ignored some of the raw input features because of their null variance. In particular, we kept 14 sensor measurements out of the total 21 sensors, whose indices are 2, 3, 4, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20 and 21, this in order to be comparable with other works using the same approach, e.g. [18] and [4].

B. Model evaluation

We are considering two objective metrics to test the performance of the model: the Scoring Function, and the Root Mean Square Error (RMSE).

1) Scoring Function: The Scoring Function was initially proposed in [12] and is defined as:

$$S = \sum_{i=1}^{N} s_i, \text{ where } s_i = \begin{cases} e^{\frac{-h_i}{13}} - 1, \; h_i < 0 \\ e^{\frac{h_i}{10}} - 1, \; h_i \geq 0 \end{cases} \tag{16}$$

where S is the computed score, N is the total number of testing data samples, and $h_i = RUL'_i - RUL_i$ is the difference between the estimated RUL and the groundtruth RUL, with respect to the i-th data point. This Scoring Function was designed to favor a safer, early prediction (i.e. estimating a smaller RUL value with respect to the groundtruth), since late prediction may result in more severe consequences.

2) Root Mean Square Error: The RMSE is a common metric to evaluate prediction accuracy of the RUL, which gives equal weights for both early and late predictions. The RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(RUL'_i - RUL_i)^2} \tag{17}$$

where RMSE is the computed RMSE value, N is the total number of testing data samples, RUL'_i and RUL_i represent respectively the estimated RUL and the groundtruth RUL, with respect to the i-th data point.

C. Model implementation

In our experiments we did not do a thorough search for the hyper-parameters of the network, as we opted for a fixed set of hyper-parameters in order to both be directly comparable to the results shown in [1] and because of resources constraints. In particular, in the aforementioned set of hyper-parameters the hidden sizes of the two LSTM networks were set to 32 and 64 respectively, while the dimension of the memory bank in the NTM was set to 64 times 32, and finally the number of neurons in the two Feedforward networks were set to 8 and 1 respectively.

We also chose an initial learning rate of 0.005, decaying it by 0.6 every 10 training epochs with a maximum amount of 50 training epochs. We are using the mini-batch gradient descent training technique with a batch size of 100, and the RMSProp algorithm [22] for optimization, with the default values for momentum and weight decay.

The LSTM states and the biases were initialized to zero, whereas the weights were sampled by using a normal distribution with mean and standard deviation values set to 0 and 0.01 respectively.

Finally, we used PyTorch 1.3.0 to implement our model.

D. Discussion of the results

The results of our experiments are shown and compared to other single-stream and double-stream architectures in Table II.

First of all, we can see that our approach, based on the combination of a Neural Turing Machine and an LSTM-based network, achieves better results than those obtained with an approach solely based on LSTM networks, which supports our claim, confirming that the usage of an external memory can better capture the hidden patterns and better encode the most informative characteristics of the data.

Furthermore, in the Table II it can also be seen that our proposed method compares really closely to [17] yet we are neither performing any pretrain, nor we are optimizing the hyper-parameters (as previously said in Section III we are keeping a fix set of hyper-parameters). The aforementioned techniques are in fact known to help when a boost to a model performance is needed, especially the pretraining stage [23]. We have not included them in our method yet, but pretraining and evolutionary techniques can be easily applied to any approach independently from its architecture, possibly leading to significant improvements. Because of this we plan to work on it in the nearest future and perform a thorough ablation study to better understand all the implications. On one hand, the study made by Ellefsen *et al.* shows that the authors obtain improvements both in Score and in RMSE using the unsupervised pretraining stage, which helps setting the initial weights near a local minimum [23]. On the other hand, the use of genetic algorithms to tune the hyper-parameters would most probably improve the performance of our model by localizing a set of favorable hyper-parameters.

The preprocessing performed in [24] tests two different values for the time window length (50 and 70), both higher than the length we used (30). They also show that they obtain better performance using 50 as the window length. In [18] the authors test two values for the window length (15 and 30), and in their case they show that a higher value for the time window length improves the obtained performance. Neither of those papers compare the effects of using 30 and 50 as the length of the window, so it is possible that a 50-steps window is more informative than a 30-steps window. Moreover, the model proposed in [24] also uses Dropout which again could be a cause of better generalization.

Although from the point of view of the RMSE we are obtaining similar values to those obtained by the CNN-based network used in [8], our Score is lower, meaning that our model seems to favor earlier (and thus safer) predictions.

Our proposed method, even though using only a single-stream architecture, obtains comparable results to those obtained by double-stream architectures. In particular, our solution obtains both a better Score and a better RMSE with respect to [18], while still using a single-stream, whereas it does not show better performance with respect to [4]. The improvements shown in [4] may be imputable to the usage of the double-stream architecture itself because, as seen in [21] and [8], the results obtained by using a single LSTM and a single CNN respectively are pretty far from the current state-of-the-art. The double-stream architecture proposed in [4] instead, by making a single LSTM and a single CNN run in parallel, achieves a much more favorable result.

Note that in this paper we explore the integration of the NTM component with the widely used LSTM networks in order to evaluate its impact. However, given the modularity of the Neural Turing Machine, our mnemonic component can be easily integrated in other competitive architectures in order to improve their performance. For example, the CNN-based approach proposed in [8] may benefit from the use of an NTM shared between the convolutional layers of the architecture. In double-stream architectures, such as [4] and [18], the NTM may be used as a "bridge" between the two streams. We leave as a future work the integration of the NTM in the aforementioned architectures.

[*] In [4] the experimental setting is different in the RMSE evaluation.

TABLE II
COMPARISON WITH THE LITERATURE. (SS) AND (DS) INDICATES WHETHER THE ARCHITECTURE IS SINGLE- OR DOUBLE-STREAM.

Methods	Years	Score	RMSE
		FD001	
MLP [7] (ss)	2016	1.80×10^4	37.56
SVR [7] (ss)	2016	1.38×10^3	20.96
RVR [7] (ss)	2016	1.50×10^3	23.80
CNN [7] (ss)	2016	1.29×10^3	18.45
LSTM [1] (ss)	2017	3.38×10^2	16.14
ELM [16] (ss)	2017	5.23×10^2	17.27
DBN [16] (ss)	2017	4.18×10^2	15.21
MODBNE [16] (ss)	2017	3.34×10^2	15.04
BLSTM [14] (ss)	2018	–	14.26
RNN [8] (ss)	2018	3.39×10^2	13.44
DCNN [8] (ss)	2018	2.74×10^2	12.61
BiLSTM [15] (ss)	2018	2.95×10^2	13.65
GADLM [17] (ss)	2019	$\mathbf{2.31 \times 10^2}$	12.56
Attn-DLSTM [24] (ss)	2019	–	**12.22**
BHLSTM [3] (ss)	2019	3.76×10^2	–
HDNN [18] (ds)	2019	2.45×10^2	13.01
DAG [4][*] (ds)	2019	2.29×10^2*	11.96*
Our approach (LSTM) (ss)	2020	3.39×10^2	16.16
Our approach (LSTM+NTM) (ss)	2020	$\mathbf{2.42 \times 10^2}$	**12.50**

About the results we obtained, it is possible to see in Figure 5 that in the majority of the mispredictions performed by our model the error (computed as $RUL'_i - RUL_i$, *i.e.* the difference between the predicted and the groundtruth RUL) is between 0 and 20, so it tends to overestimate the remaining useful life of the mechanical device. Considering the results in Figure 6 it is easily noticeable that our model fails to estimate the RUL when it is higher than 120. This may be due to the piece-wise linear labeling function we are using. In [25] it is possible to see that they predict RUL values higher than 130, but they are using a linear degradation function with respect to the cycle, and not the piece-wise function that we are using.

V. Conclusions and future work

In this paper we propose a novel approach to the Remaining Useful Life Estimation problem based on Neural Turing Machines, and in experimental section we show that we obtain very favorable and promising results considering that we are

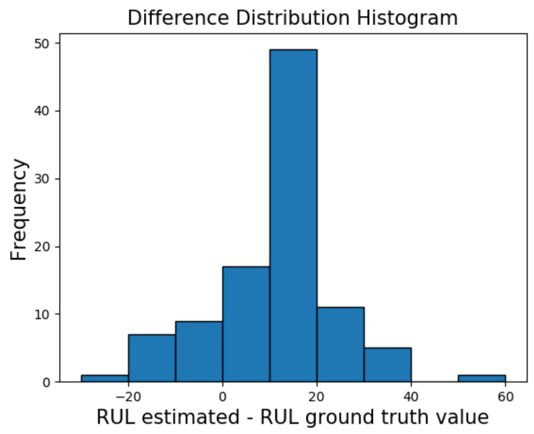

Fig. 5. Histogram of the prediction error (calculated as $RUL_i' - RUL_i$): it shows that in 46 of the 100 test series the prediction error was between 10 and 20.

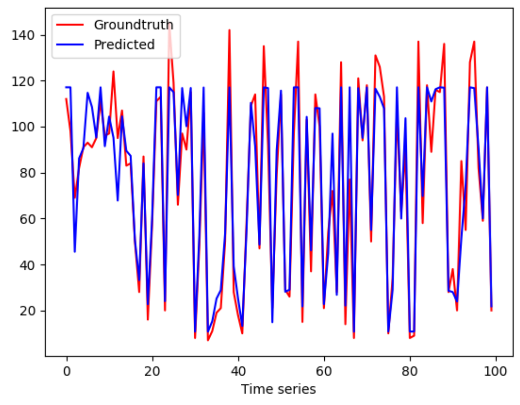

Fig. 6. Predicted RUL values and groundtruth RUL values for dataset FD001.

using a single-stream architecture.

Furthermore, being a first approach using the Neural Turing Machines for the RUL estimation problem there is room for future improvements. In particular, a first step could be finding a more favorable set of hyper-parameters by utilizing genetic algorithms, as done in [17], since the hyper-parameters used in our experiments were fixed. Another possible improvement could involve the design of a double-stream architecture revolving around the exploitation of NTMs. Moreover, we did not explore the use of other RNNs, nor the use of bidirectional RNNs. For instance, as seen in [15], the use of Bidirectional LSTMs could bring improvements with respect to the use of unidirectional LSTMs since the former can explore the sequence in input along both directions while the latter cannot. A further development could be the introduction of an Attention model in the feature extraction layers (*i.e.* in the LSTM network) in our architecture, as done in [24]. A final way to boost the performance of our model could be the usage of a different loss function, oriented to safer predictions. In particular, possible examples of such loss functions can be the Asymmetric Square Error (ASE) and the Asymmetric Absolute Error (AAE) functions introduced in [3], which are shown to make the model generate earlier predictions.

References

[1] Shuai Zheng et al. "Long short-term memory network for remaining useful life estimation". In: *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*. IEEE. 2017.

[2] Dawn An, Joo-Ho Choi, and Nam Ho Kim. "Prediction of remaining useful life under different conditions using accelerated life testing data". In: *Journal of Mechanical Science and Technology* 32.6 (2018), pp. 2497–2507.

[3] Ahmed Elsheikh, Soumaya Yacout, and Mohamed-Salah Ouali. "Bidirectional handshaking LSTM for remaining useful life prediction". In: *Neurocomputing* 323 (2019), pp. 148–156.

[4] Jialin Li, Xueyi Li, and David He. "A directed acyclic graph network combined with cnn and lstm for remaining useful life prediction". In: *IEEE Access* 7 (2019), pp. 75464–75475.

[5] Qiyao Wang et al. "Remaining useful life estimation using functional data analysis". In: *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*. IEEE. 2019.

[6] Ahmed Mosallam, Kamal Medjaher, and Noureddine Zerhouni. "Data-driven prognostic method based on Bayesian approaches for direct remaining useful life prediction". In: *Journal of Intelligent Manufacturing* 27.5 (2016), pp. 1037–1048.

[7] Giduthuri Sateesh Babu, Peilin Zhao, and Xiao-Li 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.

[8] Xiang Li, Qian Ding, and Jian-Qiao Sun. "Remaining useful life estimation in prognostics using deep convolution neural networks". In: *Reliability Engineering & System Safety* 172 (2018), pp. 1–11.

[9] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural turing machines". In: *arXiv preprint arXiv:1410.5401* (2014).

[10] Chenyou Fan et al. "Heterogeneous memory enhanced multimodal attention model for video question answering". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2019.

[11] Tkačík Jan and Pavel Kordík. "Neural turing machine for sequential learning of human mobility patterns". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2016.

[12] A. Saxena and K. Goebel. "Turbofan Engine Degradation Simulation Data Set". In: *NASA Ames Prognostics Data Repository* (2008). URL: http://ti.arc.nasa.gov/project/prognostic-data-repository.

[13] F. O. Heimes. “Recurrent neural networks for remaining useful life estimation.” In: *International Conference on Prognostics and Health Management (PHM)*. IEEE. 2008.

[14] Ansi Zhang et al. “Transfer learning with deep recurrent neural networks for remaining useful life estimation”. In: *Applied Sciences* 8.12 (2018), p. 2416.

[15] Jiujian Wang et al. “Remaining useful life estimation in prognostics using deep bidirectional lstm neural network”. In: *2018 Prognostics and System Health Management Conference (PHM-Chongqing)*. IEEE. 2018.

[16] Chong Zhang et al. “Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics”. In: *IEEE transactions on neural networks and learning systems* 28.10 (2016), pp. 2306–2318.

[17] André Listou Ellefsen et al. “Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture”. In: *Reliability Engineering & System Safety* 183 (2019), pp. 240–251.

[18] Ali Al-Dulaimi et al. “A multimodal and hybrid deep neural network model for remaining useful life estimation”. In: *Computers in Industry* 108 (2019), pp. 186–196.

[19] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[20] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[21] Yuting Wu et al. “Remaining useful life estimation of engineered systems using vanilla LSTM neural networks”. In: *Neurocomputing* 275 (2018), pp. 167–179.

[22] G. Hinton. *The RMSProp optimizer.* presented in http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf. 2014.

[23] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010.

[24] Ankit Das et al. “Deep Recurrent Architecture with Attention for Remaining Useful Life Estimation”. In: *TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*. IEEE. 2019.

[25] Khaled Akkad and David He. “A Hybrid Deep Learning Based Approach for Remaining Useful Life Estimation”. In: *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*. IEEE. 2019.