A Dual-Stream architecture based on Neural Turing Machine and Attention for the Remaining Useful Life Estimation problem

Alex Falcon[1,2], Giovanni D'Agostino[1], Giuseppe Serra[1], Giorgio Brajnik[1], and Carlo Tasso[1]

[1] *Artificial Intelligence Laboratory - Università degli Studi di Udine, Udine, Italy*
falcon.alex @ spes.uniud.it
dagostino.giovanni @ spes.uniud.it
giuseppe.serra @ uniud.it
giorgio.brajnik @ uniud.it
carlo.tasso @ uniud.it

[2] *Fondazione Bruno Kessler, Trento, Italy*

ABSTRACT

Estimating in a reliable way the Remaining Useful Life (RUL) of a mechanical component is a fundamental task in the field of Prognostics and Health Management (PHM). In recent years a greater availability of high quality sensors and easiness of data gathering gave rise to data-driven models based on deep learning for this task, which has recently seen the introduction of "dual-stream" architectures. In this paper we propose a dual-stream architecture to address the RUL estimation problem through the exploitation of a Neural Turing Machine (NTM) and a Multi-Head Attention (MHA) mechanism. The NTM is a content-based memory addressing system which gives each of the streams the ability to access to and interact with the memory and acts as a fusion technique. The MHA is an attention mechanism added as a mean for our architecture to identify the existing relations between different sensor data in order to reveal hidden patterns among them. To evaluate the performance of our model, we considered the C-MAPSS dataset, a benchmark dataset published by NASA consisting of several time series related to the life of turbofan engines. We show that our approach achieves the best prediction score (which measures the safety of the predictions) in the available literature on two of the C-MAPSS subdatasets.

1. INTRODUCTION

The estimation of the Remaining Useful Life (RUL) of a mechanical device is one of the most intensively studied problems in the field of Prognostics and Health Management (Zheng, Ristovski, Farahat, & Gupta, 2017); such problem consists in predicting whether a machine is going to have a failure and, if so, when such a failure is going to occur. Such knowledge is fundamental in order to identify the devices that are more prone to issues in the short term. Identifying such devices in advance helps scheduling their maintenance and allows containing the productivity losses caused by both their downtime and by unexpected failures (Elsheikh, Yacout, & Ouali, 2019; An, Choi, & Kim, 2018).

The approaches used in order to estimate the RUL of a mechanical device are typically divided among three categories: model-based approaches, data-driven approaches, and hybrid combinations of the former two (Lebold & Thurston, 2001; Si, Wang, Hu, & Zhou, 2011). Predictions made with model-based approaches (also called physics-based approaches) are based on the design of a physical model simulating the mechanical behaviour of the analyzed machine or individual component. Given their nature, model-based approaches are often unfeasible and too resource-consuming when the piece of equipment analyzed is complex (Q. Wang, Zheng, Farahat, Serita, & Gupta, 2019). Data-driven models base their estimation of the RUL of a mechanical device on the application of techniques such as pattern recognition and machine learning to a large amount of data (Mosallam, Medjaher, & Zerhouni, 2016; An et al., 2018; Si et al., 2011). On one hand, data-driven methods do not require an extensive knowledge of the target physical device (Eker, Camci, & Jennions, 2012; Heng, Zhang, Tan, & Mathew, 2009); on the other hand, data-driven approaches are useful only when data about run-to-failure cases of the analyzed device are available or can be easily collected (Y. Fan, Nowaczyk, & Rögnvaldsson, 2019).

In recent years, thanks to the advancements made in the field of both machine learning and deep learning, data-driven approaches based on the usage of artificial neural networks such as Convolutional Neural Networks (CNN), Long Short-Term

Memory networks (LSTM), and Recurrent Neural Networks (RNN) have became more widespread (Babu, Zhao, & Li, 2016; Zheng et al., 2017). Initially, such types of neural networks were arranged sequentially in order to develop an architecture suitable for the RUL estimation problem; such architectures are commonly referred to as "single-stream architectures". More recently though it has been shown that the performance of such architectures are surpassed by the ones obtained by architectures in which the neural networks are organized in two paths; these architectures are referred to as "dual-stream architectures" (Al-Dulaimi, Zabihi, Asif, & Mohammadi, 2019; J. Li, Li, & He, 2019). In such architectures the paths have no correlation between each other, but the output of each path affects the final prediction (J. Li, Li, & He, 2019). Dual-stream architectures were developed in order to take advantage of the different features of distinct types of neural networks by combining such features. To do so, dual-stream architectures are made of two different paths which compute different functions to learn heterogeneous hidden characteristics of the input data. In each path there can be one or more types of networks concatenated to each other; the pieces of information obtained in each path are then joined together to learn a rich data representation which is then used to estimate the RUL of the mechanical device the considered input data refers to.

In this paper, following the aforementioned advancements, we propose a novel Dual-stream architecture that includes two components: a Multi-Head Attention mechanism and a Neural Turing Machine memory module. Differently from what has been previously presented in the literature, we introduce the MHA component and we locate it before the two streams of neural networks used to estimate the RUL; such design decision was taken in order to take advantage from the fact that the MHA allows a deep learning model to jointly attend to information from different representation spaces at different positions, which in turn allows the model to identify the existing relations between the different sensors that might prove helpful in revealing hidden patterns. Moreover, to boost the mnemonic capabilities of the neural networks used in our approach, we propose to use a Neural Turing Machine, which was introduced as a working memory system and it was hence used in our architecture to provide a shared memory to the neural networks used in each of the two streams. Such neural networks can thus interact with the NTM, much like the central processor of a computer can write data on and read data from a working memory. NTMs can thus be seen as a fusion technique exploiting the information extracted from the two streams in our architecture in order to provide useful features for the subsequent RUL estimation. NTMs have previously been applied with success in research fields concerning sequence learning (Lüders, Schläger, & Risi, 2016; Greve, Jacobsen, & Risi, 2016; Jan & Kordík, 2016) and language understanding (Peng & Yao, 2015). To the best of our knowledge, the impact of NTMs applied to the field of predictive maintenance has yet to be studied.

Our major contributions can be summarized as follows:

- We propose a novel neural network dual-stream architecture for the task of RUL estimation, which integrates a Neural Turing Machine and a Multi-Head Attention. Thanks to self-attention mechanisms, the MHA helps the model to focus on the most relevant input data and to combine each of the features with the others, identifying useful relations among them.
- By exploiting the shared memory and the controller network provided by the NTM, the model is able to selectively store, manipulate, and retrieve useful information from the features extracted in both paths of the network, thus providing to the model a useful information fusion technique.
- We achieve a new state-of-the-art score for FD001 and FD003 datasets.

The following sections of the paper are organized as follows: Section 2 presents the setting of the RUL estimation problem we tackled and the tools used in our approach. Section 3 describes the proposed approach, while in Section 4 the results of our experiments are illustrated on a benchmark dataset. Finally, Section 5 concludes the paper with a discussion of the experimental results obtained and some opportunities for future work.

2. RELATED WORK AND THEORETICAL BACKGROUND

Initial machine learning approaches for the RUL estimation problem were based on architectures relying on a Multi-Layer Perceptron (MLP) and a Recurrent Neural Network (RNN) (Heimes, 2008). In more recent years approaches relying on more complex types of artificial neural network were proposed. For instance, both Babu *et al.* (Babu et al., 2016) and Li *et al.* (X. Li, Ding, & Sun, 2018) relied on architectures based on Convolutional Neural Networks. Furthermore, approaches based on the application of LSTM networks (and their Bidirectional variant) were introduced in order to take full advantage of the sequential nature of the data of the C-MAPSS dataset, as seen in (Zheng et al., 2017; A. Zhang et al., 2018; J. Wang, Wen, Yang, & Liu, 2018; Elsheikh et al., 2019). All the aforementioned approaches rely on single stream architectures.

Dual-stream architectures for RUL estimation were recently introduced by Li *et al.* and also by Al-Dulaimi *et al.* in (J. Li, Li, & He, 2019; Al-Dulaimi et al., 2019). In these architectures, the available time series of data are usually cut timewise in order to obtain shorter windows of a fixed size; such windows are then given in input to each of the streams of the architecture. In the dual-stream architecture proposed by Li *et al.* one LSTM network is placed on one of the two

stream, while the other one is constituted by a CNN followed by one pooling layer and a flatten layer; the feature vectors given in output by each stream are then combined through an element-wise sum and given in input to another LSTM followed by a Feedforward layer. The dual-stream architecture of Al-Dulaimi *et al.* relies instead on three stacked LSTM networks in the first stream and by three CNNs alternated with two pooling layers in the second one. The feature vector in the CNN path goes through a flatten layer and is then joined to the vector of the LSTM path by means of a fusion layer comprising three connected Feedforward layers. For this paper we adopted an approach based on a dual-stream architecture because in literature such architectures achieved better results than single-stream ones; moreover, we also introduce two components which impact has not been studied extensively in literature: the Multi-Head Attention and the Neural Turing Machine. Such components are respectively used in order to weigh the input data and to boost the sequence encoding performed by the neural networks used.

Neural Turing Machines (Graves, Wayne, & Danihelka, 2014) can be seen as a sequence processing model inspired by Turing Machines or the interaction between the Central Memory and the Processing Unit found in personal computers. NTMs were conceived as a way to enrich the capabilities of standard recurrent networks to simplify the solution of algorithmic tasks by means of a large addressable memory, by analog to Turing's enrichment of finite-state machines by means of an infinite memory tape. Given their ability to deal with sequential data, NTMs and neural networks augmented with NTMs have been adopted with success in many different fields, such as Video Question Answering (C. Fan et al., 2019) and Natural Language Processing (Peng & Yao, 2015).

The mechanism of Multi-Head Attention was introduced by Vaswani *et al.* in (Vaswani et al., 2017) in order to allow a model to jointly attend to information from different representation subspaces at different positions. Such mechanism exploits the concept of attention (Bahdanau, Cho, & Bengio, 2014) in a parallel fashion, giving the model the ability to attend to diverse non-overlapping subsets of the input data using multiple *heads*. This mechanism found its use in several Natural Language Processing-related fields, such as Neural Machine Translation (Vaswani et al., 2017) and Text Summarization (J. Li, Zhang, et al., 2019).

3. PROPOSED APPROACH

In our architecture (illustrated in Fig. 1), the LSTM-based path is used to extract temporal features which, for every time step, represent a summary of the evolution of the value measured by the considered sensors. Simultaneously the CNN-based path is used to extract a single local feature which is representative for the sensors at a certain time step. In our approach the sensor data is first given in input to a Multi-Head Attention (MHA) whose output is then fed into the two streams. Such design decision was made to weigh the sensor data with self-attentive scores: the MHA computes a representation of the raw sensor data based on the interaction between different projections of the same input data. This allows the neural networks in the two paths to focus on the most meaningful portions of the input.

In the rest of the paper, we identify the preprocessed input data as $S \in \mathcal{R}^{t_l \times F}$, where t_l and F are, respectively, the size of the temporal axis and the features axis. We also call h_1 and h_2, respectively, the two hidden sizes used in the architecture. We call h_c the number of features computed by each filter of the CNN module.

3.1. The Multi-Head Attention

Given the variety of physical measurements (for instance the engine temperature and intake pressure) captured by the sensors, in the proposed approach we added a module with the aim to identify the existing relations between the different sensors: the Multi-Head Attention (Vaswani et al., 2017). This mechanism is strongly based on the concept of attention (Bahdanau et al., 2014), which is used to identify a network that gives the whole architecture the ability to focus (*i.e.* to give more attention) on specific parts of the considered data. In (Vaswani et al., 2017), the attention function is applied in parallel on different subsets of the input data: after fixing the number h of attention functions (which are implemented using h different attention *heads*), the first step performed by this mechanism consists in splitting the input matrix $S \in \mathcal{R}^{t_l \times F}$ in h equal parts, obtaining $S_1, \ldots, S_h \in \mathcal{R}^{t_l \times \frac{F}{h}}$. Each of these sections will then be given in input to a different attention head, which will calculate the self-attention on it.

In more detail, the Multi-Head Attention is performed as follows:

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O \tag{1}$$

where W^O is a matrix of learnable weights in $\mathcal{R}^{F \times F}$. In our setting Q, K, and V are the same sensor data (after preprocessing), *i.e.* $Q = K = V = S \in \mathcal{R}^{t_l \times F}$, this because we want to use the MHA as a way to compute a representation of the sensor data which takes into account several projections of the same data, in order to understand the underlying relations between them.

In order to calculate the self-attention, each of the h heads takes in input a section S_i of the input split vector. In (Vaswani et al., 2017), the $head_i$ is defined as:

$$head_i(Q_i, K_i, V_i) = Attention(Q_i W_i^Q, K_i W_i^K, V_i W_i^V) \tag{2}$$

where W_i^Q, W_i^K, and W_i^V represent matrices of learnable

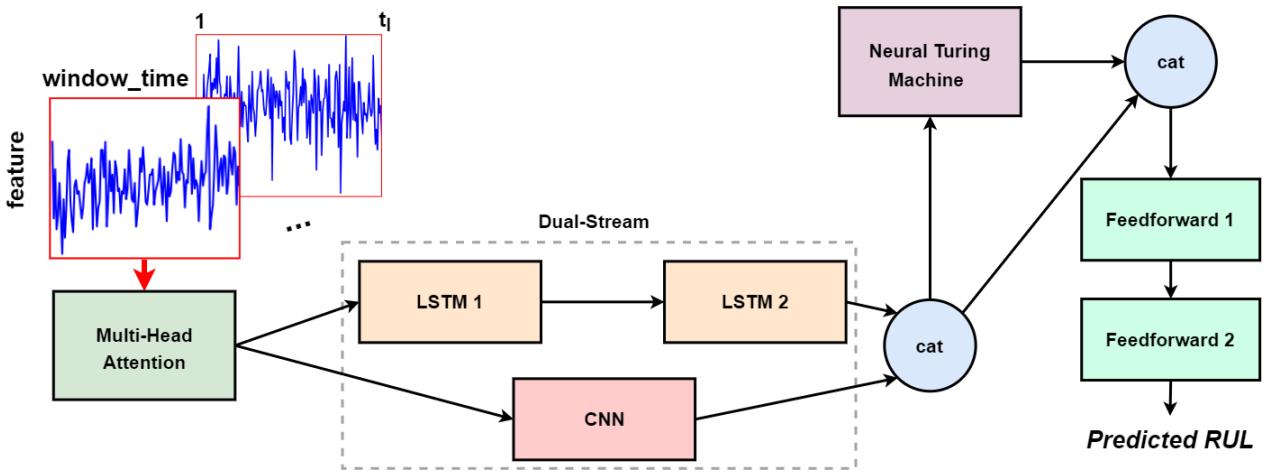

Figure 1. Graphical overview of the proposed approach. The time series are first cut into shorter windows and then fed to a MHA. The output of the Multi-Head Attention module is then given as input to the networks in each stream. The features extracted by the LSTMs of the first stream and by the CNN of the second one 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.

weights in $\mathcal{R}^{\frac{F}{h} \times h_1}$. In our case, since $Q = K = V$ and from the sensor data we obtained $S_1, \ldots, S_h$, we have that $Q_i = K_i = V_i = S_i \in \mathcal{R}^{t_l \times \frac{F}{h}}$, and $Q_i W_i^Q$, $K_i W_i^K$, $V_i W_i^V \in \mathcal{R}^{t_l \times h_1}$.

In (Vaswani et al., 2017), the *Attention* function is defined as follows:

$$Attention(q, k, v) = softmax(\frac{qk^T}{\sqrt{d_k}})v \tag{3}$$

where q, k, and v represent query, key, and value, and d_k represents the size of the key, *i.e.* the features dimension. In our case q, k, and v are three different projections of one of the sections obtained by splitting the input, *i.e.* $q = S_i W_i^Q$, $k = S_i W_i^K$, and $v = S_i W_i^V$, and $d_k = \frac{F}{h}$.

Finally, the output $S_o \in \mathcal{R}^{t_l \times F}$ of the Multi-Head Attention module is given in input to the networks in each path of the dual-stream architecture.

3.2. The Neural Turing Machine

In order to schedule the maintenance of a mechanical device, the time series that a RUL estimation model has to analyze are often constituted by long sequences of data. Solutions based on LSTM networks, despite their capability of memorizing information regarding sequences of data, might not be able to encode all the patterns found in such long time series. Because of this, in order to further boost the memory capabilities of the networks used in the proposed architecture, we coupled each stream with a shared NTM. Such choice has two consequences: first of all, 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 RUL mapping module. Secondly, the NTM acts as a fusion technique by combining the features extracted from the two heterogeneous streams, which can effectively help in the identification of the aforementioned hidden patterns. The NTM used in our approach is based on the one implemented by Fan *et al.* in (C. Fan et al., 2019), which has been customized in order to fit to our approach for 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 $M = [m_1, m_2, ..., m_{h_2}]$, where $m_i \in \mathcal{R}^{1 \times h_1}$, and a memory hidden state $h_t \in \mathcal{R}^{1 \times h_1}$ (updated at each time step), while the controller unit consists of a Feedforward network. As visible in Figure 2 the inputs to the memory at time step t are represented by the vector $o_t \in \mathcal{R}^{1 \times (h_2 + h_c)}$, which consists of the concatenation of the features extracted from each of the two streams of our architecture. The NTM supports three types of operations: write operations, read operations, and hidden state updates.

Write operation. The content to be written into the memory at time t is represented by the content vector $c_t \in \mathcal{R}^{1 \times h_1}$, in which the features extracted from the two streams are transformed and combined with the previous memory hidden state in order to maintain some of the information previously stored in the memory and introduce some of the new knowledge gathered from the input data. The content vector is computed as follows:

$$c_t = \sigma(o_t W_{oc} + h_{t-1} W_{hc} + b_c) \tag{4}$$

where $o_t \in \mathcal{R}^{1 \times (h_2 + h_c)}$ represents the current input vec-

tor, and $h_{t-1} \in \mathcal{R}^{1\times h_1}$ the previous hidden state. $W_{oc} \in \mathcal{R}^{(h_2+h_c)\times h_1}$ and $W_{hc} \in \mathcal{R}^{h_1\times h_1}$ represent trainable weights, and $b_c \in \mathcal{R}^{h_1}$ represents the bias. To write into the memory slots of the NTM, attention weights are defined as $\alpha_t = \{\alpha_{t,1}, \ldots, \alpha_{t,i}, \ldots, \alpha_{t,h_2}\}$ such that:

$$a_t = v_a \, tanh(c_t W_{ca} + h_{t-1} W_{ha} + b_a) \tag{5}$$

and

$$\alpha_{t,i} = \frac{exp(a_{t,i})}{\sum_{j=1}^{h_2} exp(a_{t,j})} \text{ for } i = 1, ..., h_2 \tag{6}$$

satisfying $\sum_i \alpha_{t,i} = 1$. $W_{ca} \in \mathcal{R}^{h_1\times h_2}$, $W_{ha} \in \mathcal{R}^{h_1\times h_2}$, $v_a \in \mathcal{R}$ representing the trainable weights, and $b_a \in \mathcal{R}^{h_2}$ representing the bias. Each memory slot m_i is then updated in the following way:

$$m_i = \alpha_{t,i} c_t + (1 - \alpha_{t,i}) m_i \text{ for } i = 1, ..., h_2 \tag{7}$$

Read Operation. The next step for the memory module is to read from the memory slots M. The normalized attention weights $\beta_t = \{\beta_{t,1}, \ldots, \beta_{t,i}, \ldots, \beta_{t,h_2}\}$ are such that:

$$b_t = v_b \, tanh(c_t W_{cb} + h_{t-1} W_{hb} + b_b) \tag{8}$$

and

$$\beta_{t,i} = \frac{exp(b_{t,i})}{\sum_{j=1}^{h_2} exp(b_{t,j})} \text{ for } i = 1, ..., h_2 \tag{9}$$

where $W_{cb} \in \mathcal{R}^{h_1\times h_2}$, $W_{hb} \in \mathcal{R}^{h_1\times h_2}$, and $v_b \in \mathcal{R}$ represent the trainable weights, and $b_b \in \mathcal{R}^{h_2}$ represents the bias. The content $r_t \in \mathcal{R}^{1\times h_1}$ read from the external memory is the weighted sum of each memory slot content:

$$r_t = \sum_{i=1}^{h_2} \beta_{t,i} \cdot m_i \tag{10}$$

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 as following:

$$h_t = \sigma(o_t W_{oh} + r_t W_{rh} + h_{t-1} W_{hh} + b_h) \tag{11}$$

where $W_{oh} \in \mathcal{R}^{(h_2+h_c)\times h_1}$, $W_{rh} \in \mathcal{R}^{h_1\times h_1}$, and $W_{hh} \in \mathcal{R}^{h_1\times h_1}$ represent trainable weights, and $b_a \in \mathcal{R}^{h_1}$ represents the bias.

3.3. The Dual-stream module

LSTM-based path. The data usually considered in the PHM field is typically composed of long time series of data measured by sensors. To model temporally-related sequential data and the evolution of its intrinsic characteristics, Recurrent Neural Networks have shown good performance in discovering hidden patterns in data. Given the sequential nature and the significant length of the time series data at hand, for

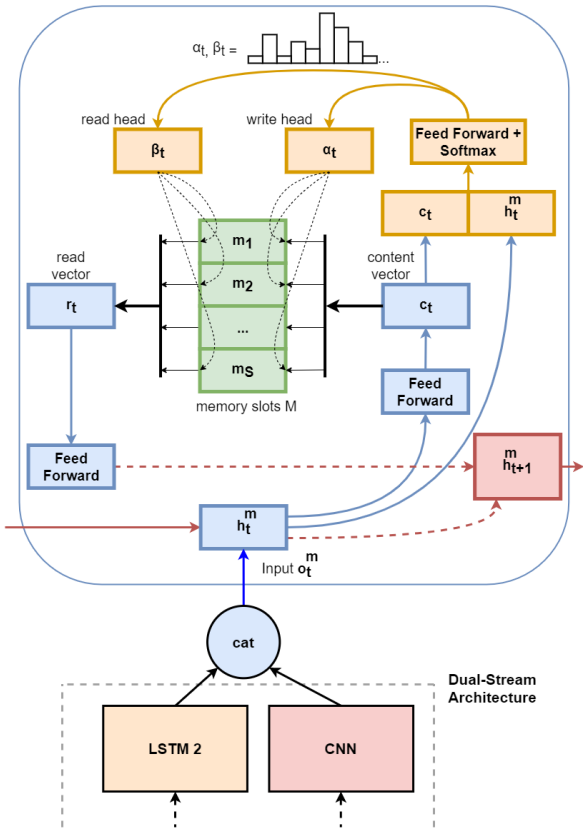

Figure 2. The external memory of our architecture at time t with memory slots $M = [m_1, m_2, ..., m_{h_2}]$, read and write heads α_t and β_t, input vector o_t, and hidden state h_t.

the first path of our architecture we exploited an LSTM-based network, similarly to what has been done in other approaches found in literature (Al-Dulaimi et al., 2019; Wu, Yuan, Dong, Lin, & Liu, 2018; J. Li, Li, & He, 2019). Such kind of neural networks were introduced in (Hochreiter & Schmidhuber, 1997) in order to mitigate the problems afflicting other kinds of RNN when handling long series of data, namely the vanishing and exploding gradients problems (Bengio, Simard, & Frasconi, 1994). Based on our preliminary experiments and given the output of the Multi-Head Attention S_o, in the first stream of our architecture we use two stacked LSTM networks. The output of this path is defined as $L_o \in \mathcal{R}^{t_l\times h_2}$ and consists of the sequence of hidden states computed by the second LSTM, given in input the hidden states of the first LSTM.

CNN-based path. In the second path of our architecture we decided to opt for a single Convolutional Neural Network; the rationale behind this choice is that, while the LSTM path focuses on and keeps track of the evolution throughout time of the sensor measurements, the CNN path focuses more on what is being measured at a certain time step t. The CNN we are using does so by computing a *single* value for each time step, which is representative of the situation at that precise

time step and it is independent of what was measured in the other time steps. The CNN takes in input a reshaped time window of size $t_l \times \frac{F}{p} \times p$ (with t_l being the size of the original time window) and outputs a feature vector of size $t_l \times 1$. This is obtained by exploiting t_l filters of size $\frac{F}{p} \times p$, while both the padding size and the stride are set to zero.

3.4. 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:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(RUL'_i - RUL_i)^2 \tag{12}$$

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

3.5. Data preprocessing

Typical time series exploited in PHM use different scales, are not labeled, and can be extremely long. To deal with this, the data is preprocessed by following three steps: by normalizing the data by means of a Min-Max feature scaling, 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.

Feature Scaling. 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, the value of a data point X of a certain feature is scaled to a new value X' through Min-Max feature scaling, which is defined as:

$$X' = \frac{2(X - X_{min})}{X_{max} - X_{min}} - 1 \tag{13}$$

where X_{min} and X_{max} represent respectively the minimum and maximum value of the feature. By doing so, all the data points vary within the range $[-1, 1]$.

RUL Target Function. To represent the remaining useful life of an engine we apply the piece-wise linear RUL target function proposed in (Heimes, 2008), which is the standard approach to deal with this problem (Babu et al., 2016; Zheng et al., 2017; Al-Dulaimi et al., 2019). In particular, we limit the maximum value of the RUL function to 125, as is done in (Al-Dulaimi et al., 2019; J. Li, Li, & He, 2019). 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 plausible model to represent the degradation of an engine, as the degradation of the analyzed system typically starts only after a certain amount of usage (Heimes, 2008; Babu et al., 2016).

Dataset	FD001	FD002	FD003	FD004
Train trajectories	100	260	100	248
Test trajectories	100	259	100	248
Operating conditions	1	6	1	6
Fault conditions	1	1	2	2

Table 1. Summary of the subdatasets of the C-MAPSS dataset.

Sliding time window. The sliding time window approach is used to extract data so that the input of the prediction model has a fixed length; our approach is in line with the ones followed in other works, such as (Al-Dulaimi et al., 2019) and (J. Li, Li, & He, 2019). The signal has F features and the signal length is L; the data is hence 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 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.

4. EXPERIMENTAL FINDINGS

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 (Saxena & Goebel, 2008). It includes four 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. A summary about the number of time series (called *trajectories*), fault conditions, and operational conditions in the datasets is available in Table 1. 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. (Al-Dulaimi et al., 2019) and (J. Li, Li, & He, 2019).

It should also be noted that datasets FD002 and FD004 involve the presence of six operational conditions, related to the value of the three operational settings. As reported in (Saxena & Goebel, 2008), these conditions need to be taken into account to have a better prediction capability since the engine behaves differently based on the condition in which it is operating. Moreover, the raw input data are normalized depending on the condition in which they have been taken. To find the condition in which a sample was taken, we used the clustering algorithm KMeans (MacQueen, 1967) with $n = 6$. In this way, the algorithm can identify and isolate the six clusters,

represented by the six possible combinations of the values of the three operational settings available in the dataset. Each sample in datasets FD002 and FD004 is augmented by six features representing the one-hot encoding of the operational condition in which the sample was taken: in such a way the total amount of features for these two datasets is 20. A similar approach was also reported in (Zheng et al., 2017). Finally, we chose to use different sliding window lengths for each dataset, based on the minimum length of the available time series: for the data of the subdataset FD001 we set the window length $t_l = 31$, for FD002 $t_l = 21$, for FD003 $t_l = 38$, and for FD004 $t_l = 19$. By doing so, we are able to obtain 17631, 48559, 21020, and 56767 windows, respectively, for FD001, FD002, FD003, and FD004.

4.1. Model performance evaluation

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

Scoring Function. The Scoring Function was initially proposed in (Saxena & Goebel, 2008) and is defined as:

$$S = \sum_{i=1}^{N} s_i, \text{ where } s_i = \begin{cases} e^{\frac{-h_i}{13}} - 1, h_i < 0 \\ e^{\frac{h_i}{10}} - 1, h_i \geq 0 \end{cases} \tag{14}$$

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.

Root Mean Square Error (RMSE). The RMSE is a metric commonly used to evaluate the prediction accuracy of the RUL of a mechanical device; such metric gives equal weights for both early and late predictions. RMSE is defined as:

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

where 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.

4.2. Model implementation

For the Multi-Head Attention, we chose to set the number of heads h to 2. Such choice was taken because, having 14 features, we could only pick between 7 and 2 heads and the latter option gave us better results in the testing phase. Similarly, there are 20 features for datasets FD002 and FD004, and again we chose to use 2 heads. For the LSTM-based path, the hidden sizes of the two LSTM networks were set to 32 and 16 respectively. The LSTM states and 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. As previously stated, for the CNN-based path to output a single value per time step, *i.e.* $h_c = 1$, we set the size of the filters to $\frac{F}{p} \times p$ with $p = 2$, which means that the filter sizes were set to 7×2 for FD001 and FD003, and to 10×2 for FD002 and FD004, while both the padding size and the stride were set to zero. Both the input and output channels were set to the length of the window size t_l, this in order to obtain a sequence of values as long as the time windows itself. The dimension of the memory bank in the NTM was set to $h_1 \times h_2 = 32 \times 16$, and finally the number of neurons in the two Feedforward networks were set to 8 and 1 respectively.

We chose an initial learning rate of 0.005, decaying it by 0.8 every 15 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 for the FD001 and FD003 subdatasets, while for FD002 and FD004 the batch size was set to 259 and 248 respectively. The RMSProp algorithm (Hinton, 2014) has been used for optimization, with the default values for momentum and weight decay. We used PyTorch 1.3.0 to implement our model.

4.3. Results discussion

The results of our experiments are shown and compared to other single-stream and double-stream architectures for RUL estimation in Table 2. Each result reported in such table is the best one among the five runs. The standard deviation σ of the results obtained in the five test runs for each dataset are: $\sigma = 2.847$ for FD001, $\sigma = 42.677$ for FD002, $\sigma = 4.210$ for FD003, and $\sigma = 67.246$ for FD004. Such results were obtained by running our model for 5 times on each of the subdatasets.

As shown in Table 2, our model learns how to handle situations with one operating condition and multiple fault conditions better than all the other models (as suggested by the score obtained on FD001 and FD003), by overcoming the other competitors by a sensible margin.

Despite the great improvements obtained in datasets FD001 and FD003, the same does not apply to situations with multiple operating conditions, that is FD002 and FD004. Yet, it must be noted that the same result can be found in all the models proposed in the literature, and not only in ours: in fact, the scores obtained on these two datasets are higher than those reported for FD001 and FD003 in all the considered architectures. A strong motivation for such lower performance may be related to the fact that FD002 and FD004 have many time series whose RUL is higher than the maximum value used to label the training data, forcing the model to make an early prediction for these time series. This not only shows that the

Methods	Year	Score			
		FD001	FD002	FD003	FD004
MLP (Babu et al., 2016) (ss)	2016	1.80×10^4	7.80×10^6	1.74×10^4	5.62×10^6
SVR (Babu et al., 2016) (ss)	2016	1.38×10^3	5.90×10^5	1.60×10^3	3.71×10^5
RVR (Babu et al., 2016) (ss)	2016	1.50×10^3	1.74×10^4	1.43×10^3	2.65×10^4
CNN (Babu et al., 2016) (ss)	2016	1.29×10^3	1.36×10^4	1.60×10^3	7.89×10^3
LSTM (Zheng et al., 2017) (ss)	2017	3.38×10^2	4.45×10^3	8.52×10^2	5.55×10^3
ELM (C. Zhang, Lim, Qin, & Tan, 2016) (ss)	2017	5.23×10^2	4.98×10^5	5.74×10^2	1.21×10^5
DBN (C. Zhang et al., 2016) (ss)	2017	4.18×10^2	9.03×10^3	4.42×10^2	7.95×10^3
MODBNE (C. Zhang et al., 2016) (ss)	2017	3.34×10^2	5.59×10^3	4.22×10^2	6.56×10^3
RNN (X. Li et al., 2018) (ss)	2018	3.39×10^2	1.43×10^4	3.47×10^2	1.43×10^4
DCNN (X. Li et al., 2018) (ss)	2018	2.74×10^2	1.04×10^4	2.84×10^2	1.25×10^4
BiLSTM (J. Wang et al., 2018) (ss)	2018	2.95×10^2	4.13×10^3	3.17×10^2	5.43×10^3
BHLSTM (Elsheikh et al., 2019) (ss)	2019	3.76×10^2	-	1.43×10^3	-
HDNN (Al-Dulaimi et al., 2019) (ds)	2019	2.45×10^2	$\mathbf{1.28 \times 10^3}$	2.87×10^2	$\mathbf{1.53 \times 10^3}$
DAG (J. Li, Li, & He, 2019)* (ds)	2019	2.29×10^2*	2.73×10^3*	5.35×10^2*	3.37×10^3*
Our approach (ds)	2020	$\mathbf{2.07 \times 10^2}$	6.03×10^3	$\mathbf{2.75 \times 10^2}$	4.80×10^3

Table 2. Comparison with the literature; (ss) and (ds) indicates whether the architecture is single- or double-stream.

scientific community is still debating which value to use to limit the RUL during training (that is, 125 in our approach and *e.g.* in (J. Li, Li, & He, 2019; Al-Dulaimi et al., 2019), or 130 in (Babu et al., 2016; A. Zhang et al., 2018)), it also shows that there is another crucial problem, that is the target RUL function.

Furthermore, we noticed that our approach favors early (and thus safer) RUL predictions. Figure 3 shows the histogram of the prediction error (calculated as the difference between the predicted RUL and the groundtruth, *i.e.* $RUL'_i - RUL_i$) on dataset FD001 and, to show that the proposed approach favors early predictions, we compare the obtained result with the histogram shown in (J. Li, Li, & He, 2019), since Li *et al.* are the only authors to report such diagram. The blue bars represent the prediction error of our architecture, while the orange ones are from (J. Li, Li, & He, 2019). The histogram shows that overall our solution proposes safer predictions: in our case there are 20 early predictions, whereas there are only 10 early predictions in (J. Li, Li, & He, 2019). Such observations seem to support the claim that giving the dual-stream architecture the ability to access to an external memory improves the safety of the model predictions. But even so, there is room for improvement; for instance, the tendency of our system to favor early predictions may be boosted by exploiting custom earliness-oriented loss function, such as the AAE and ASE loss functions proposed in (Elsheikh et al., 2019).

For completeness, in Figure 4 we also report the histograms obtained from all the four considered datasets. As previously mentioned, the histograms for FD002 and FD004 show

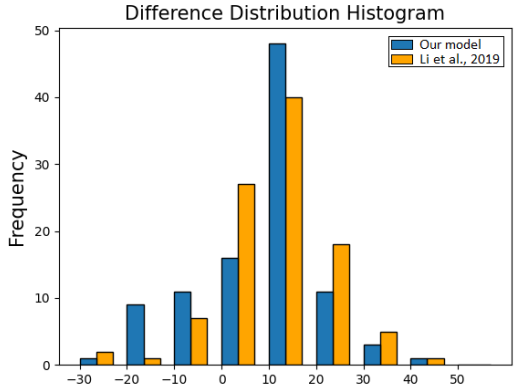

Figure 3. Histogram of the prediction error on dataset FD001, computed as *estimated RUL - groundtruth RUL*. The blue bars represent the prediction error of our architecture, while the orange ones are from the architecture proposed in (J. Li, Li, & He, 2019).

that there are many early predictions made with a considerable error, which main cause is likely related to the target function which makes it impossible for the network to predict RUL values higher than the maximum value used during training. This is likely the main reason for the performance drop obtained on datasets FD002 and FD004 not only in our approach, but in all the architectures proposed in literature; in order to establish whether this explanation is the source of such issues, further investigations will be made in future works.

* In (J. Li, Li, & He, 2019) the experimental setting is different in the RMSE evaluation.

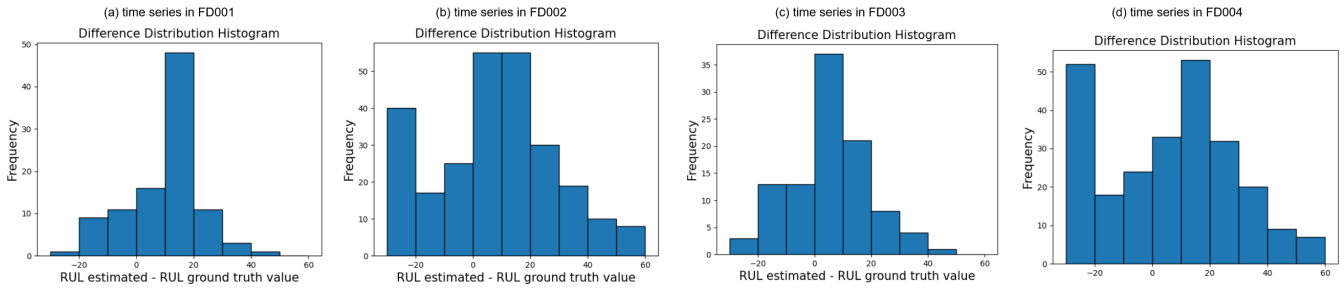

Figure 4. Histograms showing the distribution of the error (measured as estimated - groundtruth) for the four datasets.

5. CONCLUSIONS AND FUTURE WORK

In this paper we propose a novel approach to the Remaining Useful Life Estimation problem based on a dual-stream architecture preceded by a Multi-Head Attention and coupled with a Neural Turing Machine. In the experimental section we highlight how we obtain favorable results with such approach. In particular, the prediction score measured on the FD001 and FD003 datasets is the best one among the architectures our system was compared to. Furthermore, there is room for future improvements considering that our approach is the first to use both a Neural Turing Machine and a Multi-Head Attention mechanism.

ACKNOWLEDGEMENT

The dataset we used is provided by the NASA Ames Prognostics Center of Excellence (https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/).

REFERENCES

Al-Dulaimi, A., Zabihi, S., Asif, A., & Mohammadi, A. (2019). A multimodal and hybrid deep neural network model for remaining useful life estimation. *Computers in Industry*, *108*, 186–196.

An, D., Choi, J.-H., & Kim, N. H. (2018). Prediction of remaining useful life under different conditions using accelerated life testing data. *Journal of Mechanical Science and Technology*, *32*(6), 2497–2507.

Babu, G. S., Zhao, P., & Li, X.-L. (2016). Deep convolutional neural network based regression approach for estimation of remaining useful life. In *International conference on database systems for advanced applications.*

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157–166.

Eker, Ö. F., Camci, F., & Jennions, I. K. (2012). Major challenges in prognostics: study on benchmarking prognostic datasets. In *First european conference of the prognostics and health management society 2012.* PHM Society.

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

Fan, C., Zhang, X., Zhang, S., Wang, W., Zhang, C., & Huang, H. (2019). Heterogeneous memory enhanced multimodal attention model for video question answering. In *Proceedings of the ieee conference on computer vision and pattern recognition.*

Fan, Y., Nowaczyk, S., & Rögnvaldsson, T. (2019). Transfer learning for remaining useful life prediction based on consensus self-organizing models. *arXiv preprint arXiv:1909.07053*.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Greve, R. B., Jacobsen, E. J., & Risi, S. (2016). Evolving neural turing machines for reward-based learning. In *Proceedings of the genetic and evolutionary computation conference 2016.*

Heimes, F. O. (2008). Recurrent neural networks for remaining useful life estimation. In *International conference on prognostics and health management (phm).*

Heng, A., Zhang, S., Tan, A. C., & Mathew, J. (2009). Rotating machinery prognostics: State of the art, challenges and opportunities. *Mechanical systems and signal processing*, *23*(3), 724–739.

Hinton, G. (2014). *The rmsprop optimizer.* Retrieved from http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

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

Jan, T., & Kordík, P. (2016). Neural turing machine for sequential learning of human mobility patterns. In *2016 international joint conference on neural networks (ijcnn).*

Lebold, M., & Thurston, M. (2001). Open standards for condition-based maintenance and prognostic systems. In *Maintenance and reliability conference (marcon)* (Vol. 200).

Li, J., Li, X., & He, D. (2019). A directed acyclic graph net-

work combined with cnn and lstm for remaining useful life prediction. *IEEE Access*, *7*, 75464–75475.

Li, J., Zhang, C., Chen, X., Cao, Y., Liao, P., & Zhang, P. (2019). Abstractive text summarization with multi-head attention. In *2019 international joint conference on neural networks (ijcnn).*

Li, X., Ding, Q., & Sun, J.-Q. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering & System Safety*, *172*, 1–11.

Lüders, B., Schläger, M., & Risi, S. (2016). Continual learning through evolvable neural turing machines. In *Nips 2016 workshop on continual learning and deep networks (cldl 2016).*

MacQueen, J. e. a. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth berkeley symposium on mathematical statistics and probability* (Vol. 1).

Mosallam, A., Medjaher, K., & Zerhouni, N. (2016). Data-driven prognostic method based on bayesian approaches for direct remaining useful life prediction. *Journal of Intelligent Manufacturing*, *27*(5), 1037–1048.

Peng, B., & Yao, K. (2015). Recurrent neural networks with external memory for language understanding. *arXiv preprint arXiv:1506.00195*.

Saxena, A., & Goebel, K. (2008). Turbofan engine degradation simulation data set. *NASA Ames Prognostics Data Repository*. Retrieved from `https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/#turbofan`

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems.*

Wang, J., Wen, G., Yang, S., & Liu, Y. (2018). Remaining useful life estimation in prognostics using deep bidirectional lstm neural network. In *2018 prognostics and system health management conference (phm-chongqing).*

Wang, Q., Zheng, S., Farahat, A., Serita, S., & Gupta, C. (2019). Remaining useful life estimation using functional data analysis. In *2019 ieee international conference on prognostics and health management (icphm).*

Wu, Y., Yuan, M., Dong, S., Lin, L., & Liu, Y. (2018). Remaining useful life estimation of engineered systems using vanilla lstm neural networks. *Neurocomputing*, *275*, 167–179.

Zhang, A., Wang, H., Li, S., Cui, Y., Liu, Z., Yang, G., & Hu, J. (2018). Transfer learning with deep recurrent neural networks for remaining useful life estimation. *Applied Sciences*, *8*(12), 2416.

Zhang, C., Lim, P., Qin, A. K., & Tan, K. C. (2016). Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics. *IEEE transactions on neural networks and learning systems*, *28*(10), 2306–2318.

Zheng, S., Ristovski, K., Farahat, A., & Gupta, C. (2017). Long short-term memory network for remaining useful life estimation. In *2017 ieee international conference on prognostics and health management (icphm).*