

# Effective Latent Representation for Prediction of Remaining Useful Life

Qihang Wang and Gang Wu\*

Shanghai Jiaotong University, Shanghai, 200240, China

\*Corresponding Author: Gang Wu. Email: dr.wugang@sjtu.edu.cn

Received: 31 August 2020; Accepted: 19 October 2020

**Abstract:** AI approaches have been introduced to predict the remaining useful life (RUL) of a machine in modern industrial areas. To apply them well, challenges regarding the high dimension of the data space and noisy data should be met to improve model efficiency and accuracy. In this study, we propose an end-to-end model, termed ACB, for RUL predictions; it combines an autoencoder, convolutional neural network (CNN), and bidirectional long short-term memory. A new penalized root mean square error loss function is included to avoid an over-estimation of the RUL. With the CNN-based autoencoder, a high-dimensional data space can be mapped into a lower-dimensional latent space, and the noisy data can be greatly reduced. We compared ACB with five state-of-the-art models on the Commercial Modular Aero-Propulsion System Simulation dataset. Our model achieved the lowest score value on all four sub-datasets. The robustness of our model to noise is also supported by the experiments.

**Keywords:** Deep learning; predictive maintenance; remaining useful life

## 1 Introduction

Prognostic and health management (PHM) [1] is a basic requirement for condition-based maintenance in many fields where the safety, reliability, and availability of systems are considered mission-critical [2]. Remaining useful life (RUL) predictions are the jewel in the crown. The RUL of a machine is defined as the length, from the present to the end, of the useful life of one component [3]. It is necessary and valuable to predict precise RULs in many industrial areas, such as in manufacturing, aerospace, shipping, and mining, due to the fact that precise RUL predictions ensure smooth component replacements, reduce maintenance costs, and avoid permanent damage.

According to a recent review [4], the approaches of RUL predictions can be divided into 4 categories based on their basic techniques and methodologies, i.e., physics model-based approaches, statistical model-based approaches, artificial intelligence (AI) approaches, and hybrid approaches. Among them, AI approaches are highly thought of and particularly appealing due to their accuracy, robustness, and scalability in complex engineering systems [5].

Recurrent neural networks (RNNs) and convolutional neural networks (CNNs) are popular AI models in RUL predictions. The RNN is designed to model sequence data of varying lengths, which is very suitable for RUL predictions. Valine RNN has been used as well to predict the RUL [6]. However, the gradient in valine



This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

RNN tends to either explode or vanish over long time steps in the back-propagation stage, which is known as the long-term time dependence problem [7]. Long short-term memory (LSTM), one of the variants of valine RNN, has been introduced to solve this problem. Unidirectional LSTM and bidirectional LSTM (Bi-LSTM) have also been used to predict the RUL [5,8]. Besides, the CNN has shown remarkable success in various applications, especially in computer vision [9]. For example, the CNN and RNN were successfully combined in RUL predictions [10,11]. The combination of the CNN and RNN enabled the model to discover potential interrelationships and features that could significantly improve the capability of generalization.

However, directly feeding raw sensor data into those AI models might lead to poor performance because of noisy sensor data that can be contaminated due to sensor failures. Moreover, some feature engineering methods might generate a huge number of features, such as the sliding window method, which is widely used in RUL predictions. The number of features equals the size of the sliding window multiplied by the number of sensors. In real applications, one could end up with thousands of features due to the large number of sensors and complex operation conditions.

To tackle these challenges, some methods of denoising and dimensionality reduction are used. For example, Bai et al. [12] applied different kernel sizes for the CNN. Two-dimensional (2D) convolutions can extract features from several different sensors simultaneously, which sometimes reduces noisy data in the result. In contrast, one-dimensional (1D) convolutions can only function in the temporal dimension of a given sensor, and they extract features without any interference from the other sensor values. Another denoising solution is to reduce the dimensions of the data. Utilizing an autoencoder is popular in dimensionality reduction. The autoencoder is employed to learn a low-dimensional latent space for a set of high-dimensional input data. To incorporate richer representation of information in latent space, the reconstruction loss is computed between the original data (the input of encoders) and the generated data (the output of decoders). For example, an RNN-based autoencoder was built to obtain the health index for RUL predictions [13]. It maintained a set of intermediate embeddings and a set of health indices, which were computed based on normal behavior instances. Similarities were computed from the results of those sets and the test instance to predict the RUL. This method was also applied to the bidirectional RNN autoencoder to convert high-dimensional sensor data into a low-dimension embedding [14].

In practice, these methods still have some problems. Reference [13] maintained two sets and computed the similarity of every unit within each set, which is computationally expensive in the prediction phase. Meanwhile, an RNN-based autoencoder presented by Yu et al. [14] makes parallel computing impossible and the whole process computationally expensive.

In this study, we propose an end-to-end model combining an autoencoder, CNN, and Bi-LSTM (and thus we name it ACB). Our model efficiently catches both long-term features and short-term ones from the dimensionally reduced latent space. Specifically, multivariate sensor data are fed into a stacked CNN-based autoencoder to map the high dimension of data space into a lower dimension of latent space. Then, short-term dependencies are extracted by 1D convolutional layers, and long-term dependencies are extracted by subsequent Bi-LSTM layers.

To penalize the predicted RUL, which can be greater than the ground-truth RUL, and to avoid severe consequences fields such as aerospace industries [15], we propose a new custom loss function named the penalized root mean square error (P-RMSE) loss. Through employing this new function, the total loss consisting of the weighted sum of the reconstruction loss and the P-RMSE loss can be minimized.

The major contributions of this work are summarized as follows.

- a) We propose an end-to-end model ACB for RUL predictions; it combines a 1D CNN-based autoencoder, CNN layers, and Bi-LSTM layers for RUL predictions. ACB performs well in complex operations and conditions (refer to Sections 3.2 and 3.3).

- b) We propose P-RMSE, a new loss function, which penalizes the predicted RUL that is greater than the ground-truth RUL (refer to Section 3.4).

For evaluation, we conducted experiments based on the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dataset [16]. The proposed ACB method achieved RMSEs of 12.65, 14.69, 12.39, and 15.62 and score values [15] of 219.18, 1163.45, 225.04, and 1055.32 on four sub-datasets. Furthermore, ACB was compared to five AI-based prognostic baselines [10,17–20]. The results show that ACB achieved the lowest score on all four sub-datasets in the C-MAPSS dataset (refer to Section 4.5). Additionally, ACB achieved a score standard deviation of 229.24 and an RMSE standard deviation of 0.93 at different noise levels on the FD004 sub-dataset (refer to Section 4.6).

The rest of this paper is organized as follows. Section 2 describes the problem formulation and preliminaries. Section 3 elaborates on the details of ACB in RUL predictions. The environmental and implementation details are given in Section 4, along with an evaluation and analysis of ACB. Conclusions are drawn and further work presented in Section 5.

## 2 Problem Formulation

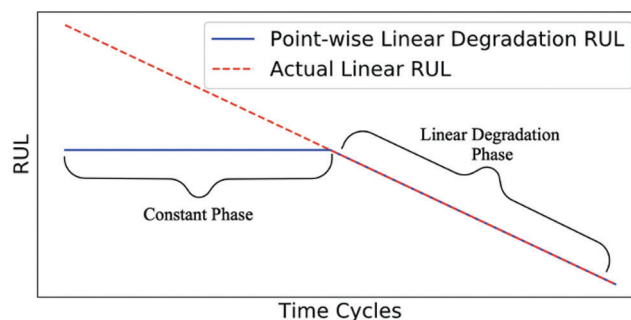
We assume that we have a system with several components (e.g., devices, equipment). Each component has  $M$  sensors equipped. In general, those sensors collect the component information, such as vibration, temperature, and electric current.

In the training phase, a training sample is a multivariate time series  $X_i \in \mathbb{R}^{T_i \times M}$ , where  $T_i$  represents the total number of time steps (also called lifetime) for the  $i$ -th component.  $X_i^t \in \mathbb{R}^M$  denotes the data collected by all  $M$  sensors of  $t$ -th time steps of the  $i$ th component, where  $t \in \{1, 2, \dots, T_i\}$ . Hence, the training data are defined as  $\mathcal{X} = \{X_i^t | i = 1, 2, \dots, N; t = 1, 2, \dots, T_i\}$ .

In the testing phase, a test sample of the  $j$ -th component is another multivariate time series  $C_j \in \mathbb{R}^{F_j \times M}$ , where  $F_j$  denotes the total number of time steps for the  $j$ -th component. Compared with the training data, test data will not contain all time steps data collected by  $M$  sensors from runtime to failure. The failure point of the  $j$ -th component is usually later than  $F_j$ , which can be defined as  $\vec{F}_j$ . The goal is to predict the remaining useful life  $\vec{F}_j - F_j$  of the  $j$ -th component based on the sensor data from time step 1 to  $F_j$ . Hence, the test data are defined as  $\mathcal{C} = \{C_j^t | j = 1, 2, \dots, L; t = 1, 2, \dots, F_j\}$ .

### 2.1 Point-Wise Linear Degradation

In practical applications, premature predictions are useless, while predictions made close enough to the failure point are valuable. In addition, the degradation of a component is negligible at the beginning of use [20]. Therefore, a point-wise linear degradation model was proposed by Babu et al. [17] and has been widely used since [8,21]. The point-wise linear degradation model hypothesizes that the target RUL consists of two phases: a constant phase and a linear degradation phase (Fig. 1). The predicted RUL is limited to a constant value to avoid overestimation, and the start point of the linear degradation phase is different for each dataset.



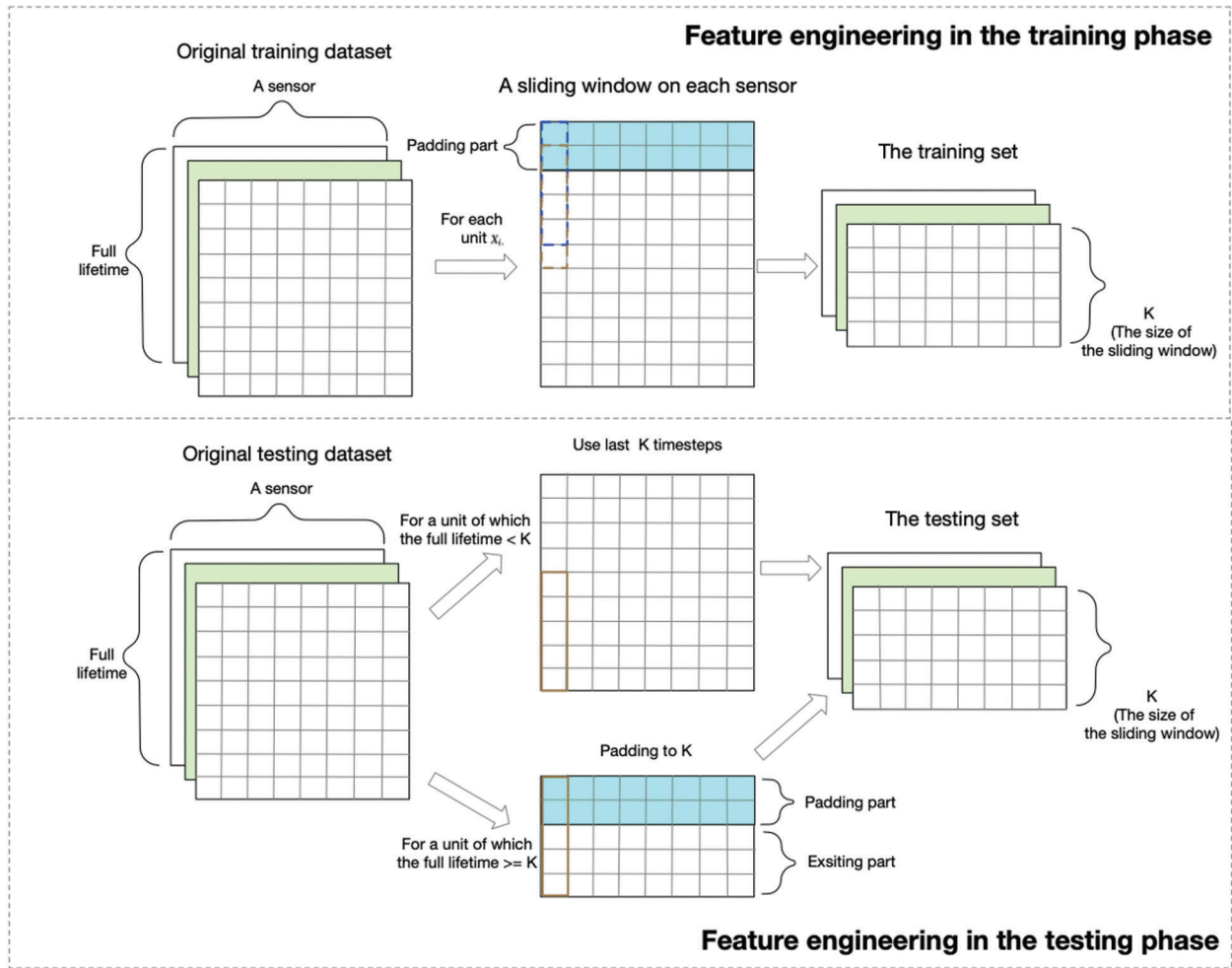
**Figure 1:** Illustration of point-wise linear degradation proposed by [17]

### 3 Method

In this section, we present ACB for RUL predictions. A feature engineering method is introduced in Section 3.1 before feeding data into our model. We elaborate on the details of the model in Sections 3.2 and 3.3. The P-RMSE is designed in Section 3.4.

#### 3.1 Feature Engineering

Before training data are fed into ACB, feature engineering is conducted to improve the performance, as shown in Fig. 2. Both the sliding window method and the padding method are used in the training phase and the testing phase. Suppose  $K$  is the sliding window's size, and  $P$  is the padding size.



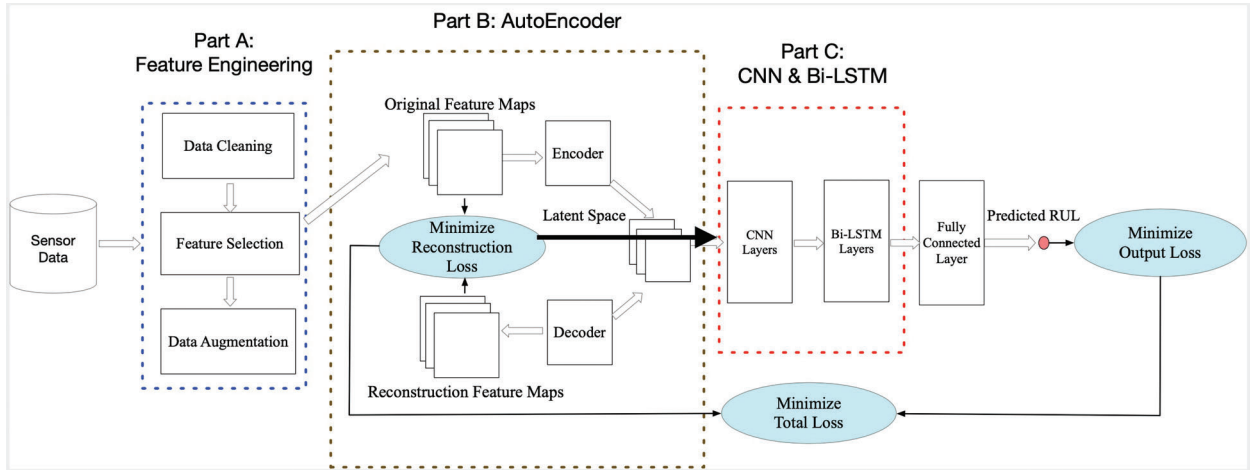
**Figure 2:** Feature engineering ( $K$  equals 5, and the size of the padding part equals 2)

In the training phase, for each unit  $X_i$ , padding 0 at the start is used to make the distribution of training data and testing data closer. Then, a window slides from  $X_i^{1-P}$  to  $X_i^{T_i-K}$ , which generates  $T_i - K + P$  slices. The remaining useful life of each slice  $[X_i^k, X_i^{k+K}]$  is computed as  $T_i - k - K$ , which can be regarded as the total number of time steps minus the end time steps of this slice. Suppose  $S_k \in \mathbb{R}^{K \times M}$  represents the slice  $[X_i^k, X_i^{k+K}]$ , and  $Y_k \in \mathbb{R}$  is the remaining useful life. Hence, the training set is defined as  $(\mathcal{S}, \mathcal{Y}) = \{(S_k, \mathcal{Y}_k) | k = 1, 2, \dots, \hat{N}\}$ , where  $\hat{N}$  denotes the size of the training set.

In the testing phase, we handle each unit based on its lifetime. Different from the training phase, for each unit, we use the last  $K$  time steps if its lifetime is greater than  $K$ ; otherwise, we pad the lifetime to  $K$ .

### 3.2 Effective Latent Representation

As shown in Part B of Fig. 3, for a given time series  $S_k \in \mathbb{R}^{K \times M}$  of the  $k$ -th training data, the auto encoder automatically extracts features and casts  $S_k$  into a low-dimension latent space  $Z_k \in \mathbb{R}^{K \times D}$ , where  $Z_k$  denotes the latent space of  $S_k$ , and  $D$  denotes the dimension of the latent space.  $Z_k$  will be used in the downstream layers. Then,  $Z_k$  is reconstructed as  $\tilde{S}_k \in \mathbb{R}^{K \times M}$ , where  $\tilde{S}_k$  represents the time series after the reconstruction in the original space. This process can be defined as



**Figure 3:** Structure of ACB

$$Z_k = g(S_k), k = 1, 2, \dots, \hat{N} \quad (1)$$

$$\tilde{S}_k = f(Z_k), k = 1, 2, \dots, \hat{N}, \quad (2)$$

where  $f(\cdot)$  and  $g(\cdot)$  denote the decoder and encoder, respectively, and can be composed of neural networks.

In order to achieve a more compact and accurate representation, the reconstructed  $\tilde{S}_k$  should be as close as possible to  $S_k$  for each  $k = 1, 2, \dots, \hat{N}$ . Suppose  $\mathcal{L}_{\mathcal{R}}(\mathcal{S}, \tilde{\mathcal{S}})$  is the reconstruction loss function. The RMSE is used for the reconstruction loss function, which is given by

$$\mathcal{L}_{\mathcal{R}}(\mathcal{S}, \tilde{\mathcal{S}}) = \sqrt{\frac{1}{\hat{N}} \sum_{k=1}^{\hat{N}} (S_k - \tilde{S}_k)^2}. \quad (3)$$

### 3.3 Temporal Convolutional Layer and Bidirectional LSTM Layer

To extract both short-term features and long-term features, convolutional layers and bidirectional LSTM layer are used, as shown in Part C of Fig. 3. Consider a given time series  $Z_k \in \mathbb{R}^{K \times D}$  of  $k$ -th training data from latent space generated by the auto encoder mentioned above.

The convolutional layer slides the filters over the input time series to generate feature maps, and this can be computed by:

$$H_k \in \mathbb{R}^{K \times F} = a(Z_k * w + b), \quad (4)$$

where  $*$  refers to the convolution operator,  $w$  and  $b$  denote the 1D weight kernel and the bias, respectively, and  $a$  denotes the non-linear activation function.  $F$  is the number of feature maps, which is usually larger than  $D$ .

Bidirectional structures have proven to be very successful in some tasks, such as natural language processing. Accordingly, time-series data are also suitable for bidirectional models. The core idea behind Bi-LSTM lies in the fact that two separate hidden layers are utilized to process the sequence of time-series data in two directions (i.e., forward and backward) to capture the information of both the past and the future. Consider a given time series  $H_k^i \in \mathbb{R}^F$  of the  $i$ -th time step of the  $k$ -th training data. Two different symbols, i.e.,  $\overrightarrow{\quad}$  and  $\overleftarrow{\quad}$ , are used to represent the forward and the backward processes. Eq. (5) describes the structure of Bi-LSTM:

$$\begin{aligned} \overleftarrow{O}_k^i &= \overleftarrow{l} \left( H_k; \overleftarrow{O}_k^{i-1}; \overleftarrow{\Theta} \right) \\ \overrightarrow{O}_k^i &= \overrightarrow{l} \left( H_k; \overrightarrow{O}_k^{i-1}; \overrightarrow{\Theta} \right) \\ O_k^i \in \mathbb{R}^B &= \left[ \overleftarrow{O}_k^i, \overrightarrow{O}_k^i \right]. \end{aligned} \quad (5)$$

Here,  $\overleftarrow{O}_k^i$  and  $\overrightarrow{O}_k^i$  are forward and backward LSTM networks, respectively.  $\overleftarrow{\Theta}$  and  $\overrightarrow{\Theta}$  denote the parameters of  $\overleftarrow{O}_k^i$  and  $\overrightarrow{O}_k^i$ .  $\overleftarrow{l}$  and  $\overrightarrow{l}$  are LSTM cells.  $B$  denotes the number of cells in Bi-LSTM.

Finally, a fully connected layer is used to generate the predicted RUL is given by:

$$\hat{Y}_k = W_o \cdot p(O_k^i), \quad (6)$$

where  $\hat{Y}_k$  indicates the predicted RUL of the  $k$ -th training data,  $W_o$  denotes the weight vector of the final fully connected layer, and  $p$  is the global average pooling layer.

### 3.4 Loss Function

In the training process, the total loss function can be divided into two parts, reconstruction loss and output loss, and is given by:

$$\mathcal{L} = \beta_1 \cdot \mathcal{L}_{\mathcal{R}}(\mathcal{S}, \tilde{\mathcal{S}}) + \beta_2 \cdot \mathcal{L}_{\mathcal{O}}(\mathcal{Y}, \tilde{\mathcal{Y}}), \quad (7)$$

where  $\beta_1$  and  $\beta_2$  are the weights of the losses. The reconstruction loss has been computed in Eq. (3). To penalize the predicted RUL that is greater than the ground-truth RUL, a new P-RMSE loss is proposed and defined as follows:

$$\mathcal{L}_{\mathcal{O}}(\mathcal{Y}, \tilde{\mathcal{Y}}) = \sqrt{\sum_{k=1}^{\hat{N}} h(\hat{Y}_k, Y_k)}, \quad (8)$$

where  $\hat{Y}_k$  denotes the predicted RUL while  $Y_k$  denotes the ground-truth RUL.  $h(\hat{Y}_k, Y_k)$  is given by:

$$h(\hat{Y}_k, Y_k) = \begin{cases} (1 + \lambda)(\hat{Y}_k - Y_k)^2 \\ (Y_k - \hat{Y}_k)^2 \end{cases}, \quad (9)$$

where  $\lambda$  is the penalty term.



## 4 Experiments

In this section, some experiments are conducted on the publicly available C-MAPSS dataset for RUL predictions. The dataset descriptions are given in Section 4.1. Then, we employ RMSE, and the score function as metrics for performance evaluation. Next, we introduce the details of the ACB implementation. Then we use the grid search method to generate the best hyper-parameters and compare our approach with other baselines approaches. Furthermore, we evaluate the proposed model's robustness to noise by adding random Gaussian noise.

### 4.1 C-MAPSS Dataset

The C-MAPSS dataset is widely used in RUL predictions as a benchmark dataset [22]. The C-MAPSS dataset is divided into four sub-datasets (from FD001 to FD004), each of which can be further divided into a training set and a testing set with different numbers of trajectories and conditions, as shown in Tab. 1. The complexity of the sub-datasets will increase with the number of operating conditions and fault conditions [23]. FD002 and FD004 are two relatively complex sub-datasets. The data are also contaminated with sensor noises.

**Table 1:** C-MAPSS dataset details from [16]

Dataset	FD001	FD002	FD003	FD004
Training trajectories	100	260	100	249
Testing trajectories	100	259	100	248
Operating conditions	1	6	1	6
Fault conditions	1	1	2	2

In each sub-dataset, data can be regarded as a 2D matrix stored in chronological order. Each row is a snapshot of information data of the current cycle. The 1st column is the engine ID, the 2nd column represents the operational cycle number, the 3rd to 5th columns list the three operating settings that have a substantial effect on the engine performance [24], and the last 21 columns present sensor values. In the training dataset, data are provided from the start to the failure point. In the testing dataset, data are provided from the start to a certain time point before the failure point.

### 4.2 Performance Metrics

In order to evaluate the performance, the RMSE is given by:

$$RMSE = \sqrt{\frac{1}{N'} \sum_{j=1}^{N'} (Y_j - \hat{Y}_j)^2}, \quad (10)$$

where  $N'$  denotes the total number of data after the augmentation,  $Y_j$  denotes the ground-truth RUL, and  $\hat{Y}_j$  denotes the predicted RUL.

In PHM area, the score function [15] is used as follows

$$\text{Score} = \begin{cases} \sum_{j=1}^{N'} \exp\left(-\frac{\hat{Y}_j - Y_j}{13}\right) & \hat{Y}_j - Y_j < 0 \\ \sum_{j=1}^{N'} \exp\left(-\frac{Y_j - \hat{Y}_j}{10}\right) & \hat{Y}_j - Y_j \geq 0 \end{cases}. \quad (11)$$

The score function was used by many studies to evaluate RUL predictions performance [25,26] and was also employed in the International Conference on Prognostics and Health Management Data Challenge. Similar to the P-RMSE loss function in Eq. (7), Eq. (11) penalizes the predicted RUL that is greater than the ground-truth RUL, and is controlled by two constants, 10 and 13.

As shown in Fig. 4, the defect of the score function is that if one predicted RUL is much larger than the ground-truth RUL, the score might be excessively large because an exponential operation is used in this function. However, the score function contains more information than the RMSE in RUL predictions.

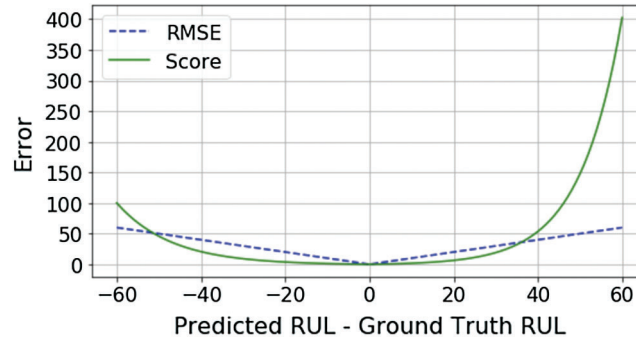


Figure 4: Comparison of RMSE and score

### 4.3 Implementation Details

We implemented ACB using the deep learning library Keras. The first two layers of the encoder consist of two 1D convolution layers with a  $3 \times 3$  kernel size and 8-dimensional hidden states, followed by a 1D max pooling layer and a batch normalization layer. A decoder is a mirrored representation of an encoder. The softmax activation function is used in the last 1D convolution layer of the decoder. The CNN layers consist of two 1D convolution layers with 12-dimensional hidden states and a dropout layer with a 0.2 dropout rate. Meanwhile, the Bi-LSTM layer is made up of 12-dimensional hidden states and a dropout layer with a 0.2 dropout rate. A 1D max pooling layer and a batch normalization layer are also used after the CNN layers and Bi-LSTM layer. Finally, a global average pooling layer and a fully connected layer are used to generate the RUL.

### 4.4 Evaluation of Hyper-Parameters

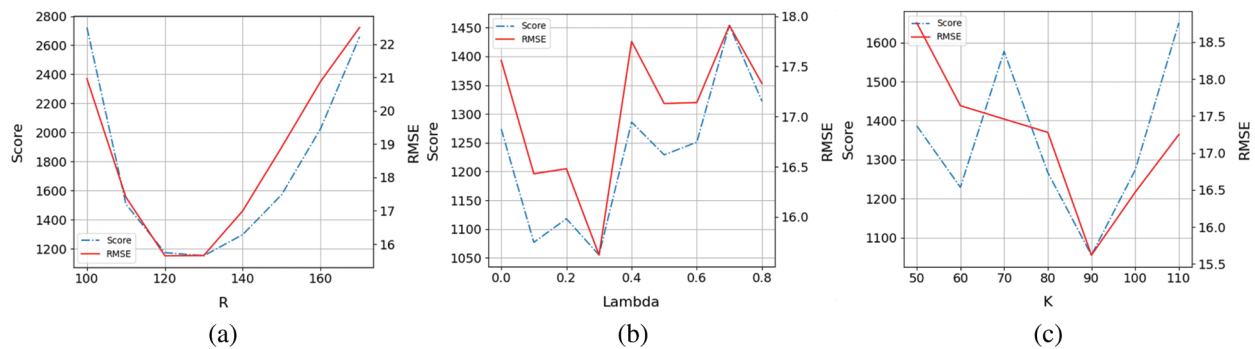
Our proposed method ACB is significantly influenced by five hyper-parameters: the size of the sliding window  $K$ , the penalty term  $\lambda$ , the reconstruction loss weight  $\beta_1$ , the output loss weight  $\beta_2$ , and the start point of linear degradation  $R$ . We set  $\beta_1$  and  $\beta_2$  to 1 as default. Meanwhile, we use the grid search method to find the best values of  $K$ ,  $\lambda$ , and  $R$  on the FD004 sub-dataset.

We increased  $\lambda$  from 0 to 0.8,  $K$  from 50 to 110, and  $R$  from 100 to 170. Therefore, the search space is  $9 * 7 * 8 = 504$ . We traversed the entire search space to find the best set of parameters  $(\hat{\lambda}, \hat{k}, \hat{R})$ . Because the search space is too large to visualize, for better display, we kept two of the parameters unchanged and then



only changed the remaining one. For example, we fixed  $(\hat{\lambda}, \hat{k})$  and then made  $R$  change from  $R_1$  to  $R_n$ , where  $n$  is the search range of parameter  $R$ .

In Fig. 5a, when  $R$  equals 130, the RMSE and score values are both the lowest. However, since the  $R$  value restricts the upper limit of the predicted RUL, overly large  $R$  values might induce high variance, while too small  $R$  values might lead to a high bias. We conduct the same experiments on all four sub-datasets. The conclusion is that different datasets might prefer different  $R$  values because of the different operating conditions. In a single-operations dataset like FD001 and FD003, setting  $R$  to 120 is most suitable; meanwhile,  $R$  equaling 130 is more reasonable on multi-operations datasets such as FD002 and FD004.



**Figure 5:** Effects of the three key hyper-parameters on performance on the FD004 sub-dataset. (a) Effects on  $R$ , (b) Effects on  $\lambda$ , (c) Effects on  $K$

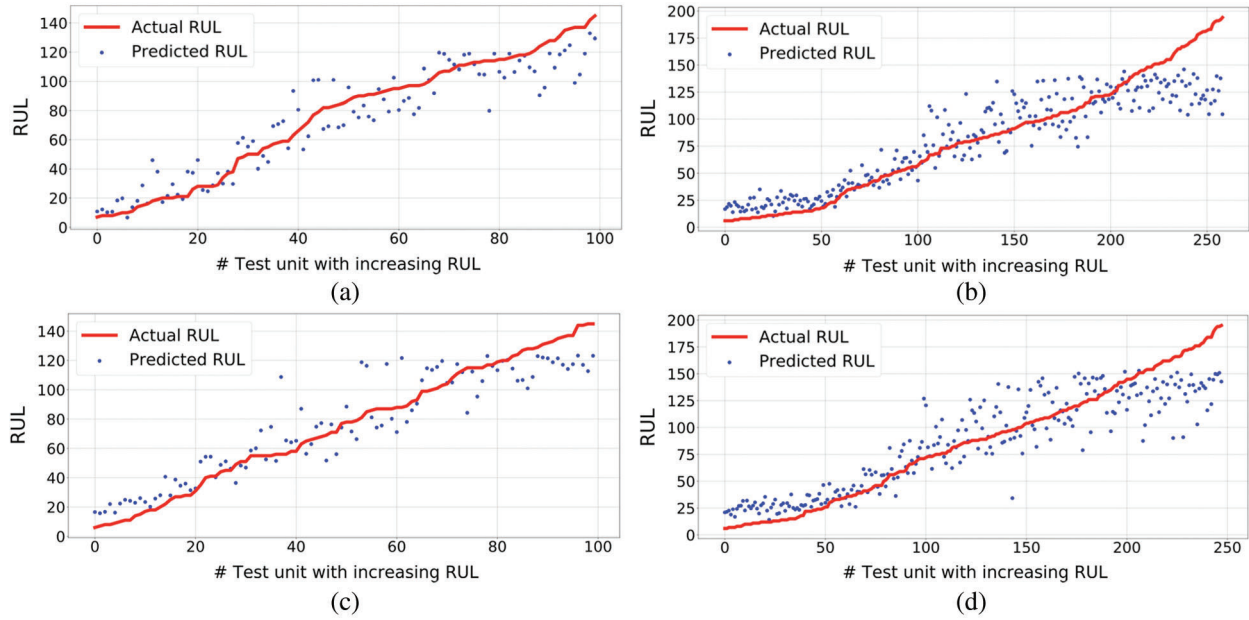
Note that in Fig. 5b, the curve has some fluctuations, but there is still an optimum when  $\lambda$  equals 0.3. Note that when  $\lambda$  equals 0, the P-RMSE is equivalent to the RMSE. A too-large penalty term  $\lambda$  results in over-penalization, which makes the predicted RUL less than the actual RUL. Hence, a relatively small  $\lambda$  is recommended.

Fig. 5c shows the effects of the size of the sliding window  $K$ .  $K$  indicates that how much information will be fed into models. However, excessive information or features might cause overfitting, while insufficient information might cause underfitting.  $K$  equaling 90 is suitable in our experiments.

#### 4.5 Experimental Results and Evaluation

ACB's predicted RULs on each sub-dataset are shown in Fig. 6 with suitable hyper-parameters as mentioned in Section 4.4. In Fig. 6, test units are sorted in ascending order by the actual RUL value for the four testing datasets. Note that at the beginning of use (on the right of the figure), the degradation of a component is negligible, which results in a weaker prediction performance. Conversely, close to the left of the figure, the component is already degraded, and thus the prediction performance is significantly improved.

We also compared ACB with several state-of-the-art baselines including the CNN [17], Deep LSTM [20], Temporal Convolutional Memory Networks (Temporal CMN [10]), Light Gradient Boosting Machine (LightGBM [18]), and Convolutional Neural Network-eXtreme Gradient Boosting (CNN-XGB [19]). The Deep LSTM employs multi-layer LSTM and a multi-layer fully connected layer, which makes full use of the sensor sequence information. Temporal CMN incorporates temporal convolutions with both long-term and short-term time dependencies. Meanwhile, LightGBM uses a light gradient boosting machine to estimate the RUL, while CNN-XGB ensembles a 1\*1 filter kernel and XGBoost to predict the RUL. The results are shown in Tab. 2.



**Figure 6:** Result visualizations of ACB on the C-MAPSS dataset (a) FD001 visualization, (b) FD002 visualization, (c) FD003 visualization, (d) FD004 visualization

**Table 2:** Performance comparisons with baselines (results of baselines come from their respective papers)

Method	Metric	FD001	FD002	FD003	FD004
CNN [17]	RMSE	18.45	30.29	19.82	29.16
	Score	1286.7	13570.0	1596.2	7886.4
Temporal CMN [10]	RMSE	23.57	20.45	21.17	21.03
	Score	1220.0	310.0	1300	4000.0
Deep LSTM [20]	RMSE	16.14	24.49	16.18	28.17
	Score	338.0	4450.0	852.0	5550.0
LightGBM [18]	RMSE	16.25	26.49	16.57	27.27
	Score	387.4	8987.1	665.4	5133.4
CNN-XGB [19]	RMSE	<b>12.61</b>	19.61	13.01	19.41
	Score	224.73	2525.99	279.36	2930.65
ACB	RMSE	12.65	<b>14.69</b>	<b>12.39</b>	<b>15.62</b>
	Score	<b>219.18</b>	<b>1163.45</b>	<b>225.04</b>	<b>1055.32</b>

To better analyze these results, Fig. 7 shows a randomly selected unit from each sub-dataset. Similar to Fig. 6, the closer the component to the failure point, the more accurate the ACB's results.

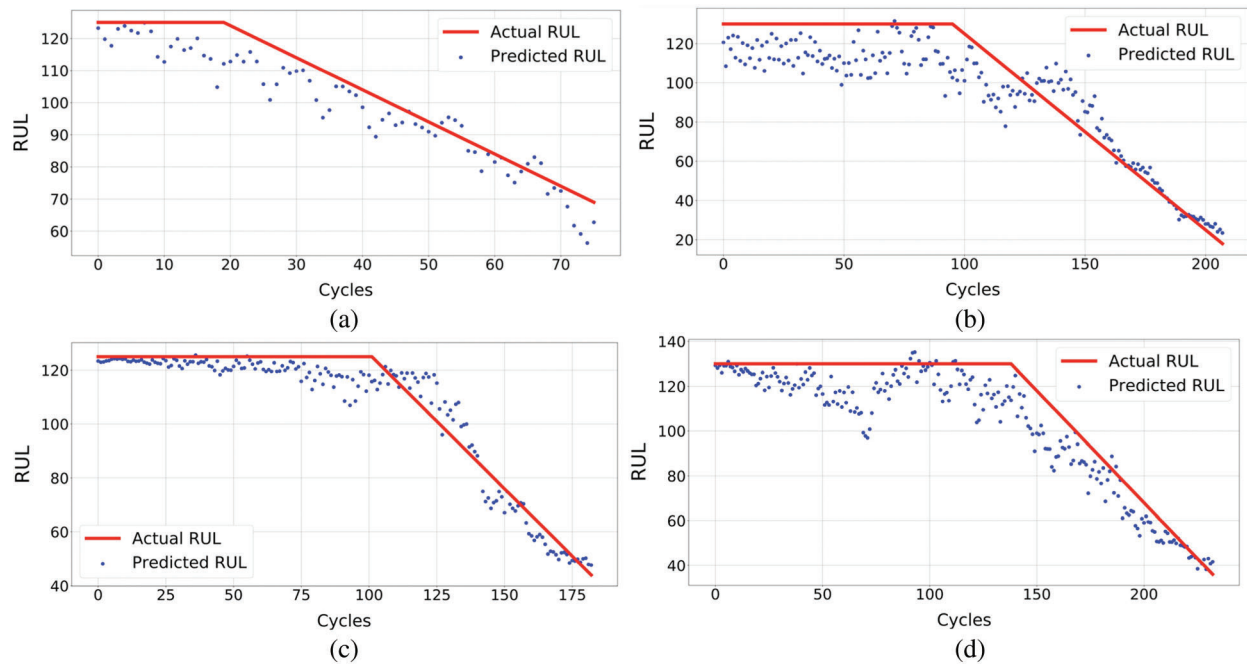
#### 4.6 Robustness to Noise

To evaluate the robustness, we evaluated the results obtained under different noise conditions. We used the method proposed by Gugulothu et al. [13] to add noise to the FD004 sub-dataset due to its complex conditions. Each testing data  $C_j \in \mathbb{R}^{F_j \times M}$  in the FD004 sub-dataset, where  $F_j$  denotes the number of time

steps for the  $j$ -th component, and  $M$  denotes the number of sensors. At each time step  $t$ ,  $C_j^t$  was corrupted with additive Gaussian noise, which generates a new version of the testing data as follows:

$$C_j^{t'} \text{ s.t. } C_j^{t'} | C_j^t \sim \mathcal{N}(C_j^t, \sigma^2 I). \quad (12)$$

As shown in Tab. 3, ACB is stable and performs well at different noise levels  $\sigma$ .



**Figure 7:** Results visualization of ACB on the C-MAPSS dataset. (a) Testing unit #3 in FD001, (b) Testing unit #1 in FD002, (c) Testing unit #1 in FD003, (d) Testing unit #35 in FD004

**Table 3:** Robustness evaluation at different noise levels  $\sigma$  on the FD004 sub-dataset

Nosie ( $\sigma$ )	Score	RMSE
0.0	1055.32	15.62
0.1	1297.53	16.12
0.2	1554.18	17.19
0.3	1640.97	18.02
Mean	1386.50	16.73
Std	229.24	0.93

## 5 Conclusions

In this work, an end-to-end deep learning network combining an autoencoder, CNN, and Bi-LSTM for remaining useful life predictions was presented. The autoencoder is designed for dimensionality reduction to adapt multivariate sensor data and noisy conditions. Meanwhile, the CNN and Bi-LSTM are used for catching short-term dependencies and long-term dependencies to improve the accuracy and efficiency.

Furthermore, we proposed the P-RMSE loss function to penalize the predicted RUL that is greater than the ground-truth RUL in order to avoid overestimation. ACB achieved promising results on the C-MAPSS dataset, and in some complex sub-datasets it performed better than other models. The robustness of ACB to noise was evaluated in experiments. In the future, we plan to optimize the LSTM structure and consider using some variants of the autoencoder, such as stacked denoising autoencoders and variational autoencoders, to prompt further model improvement.

**Funding Statement:** The authors received no specific funding for this study.

**Conflicts of Interest:** The authors declare that they have no conflicts of interest to report regarding the present study.

## References

- [1] M. Dong and D. He, "Hidden semi-markov model-based methodology for multi-sensor equipment health diagnosis and prognosis," *European Journal of Operational Research*, vol. 178, no. 3, pp. 858–878, 2007.
- [2] M. Dong and Y. Peng, "Equipment PHM using non-stationary segmental hidden semi-Markov model," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 3, pp. 581–590, 2011.
- [3] X. Si, W. Wang, C. Hu and D. Zhou, "Remaining useful life estimation—a review on the statistical data driven approaches," *European Journal of Operational Research*, vol. 213, no. 1, pp. 1–14, 2011.
- [4] Y. Lei, N. Li, L. Guo, N. Li, T. Yan *et al.*, "Machinery health prognostics: A systematic review from data acquisition to RUL prediction," *Mechanical Systems and Signal Processing*, vol. 104, pp. 799–834, 2018.
- [5] C. G. Huang, H. Z. Huang and Y. F. Li, "A bidirectional LSTM prognostics method under multiple operational conditions," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 11, pp. 8792–8802, 2019.
- [6] L. Guo, N. Li, F. Jia, Y. Lei and J. Lin, "A recurrent neural network based health indicator for remaining useful life prediction of bearings," *Neurocomputing*, vol. 240, pp. 98–109, 2017.
- [7] Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.
- [8] Y. Wu, M. Yuan, S. Dong, L. Lin and Y. Liu, "Remaining useful life estimation of engineered systems using vanilla LSTM neural networks," *Neurocomputing*, vol. 275, pp. 167–179, 2018.
- [9] X. Li, Q. Ding and J. Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering & System Safety*, vol. 172, pp. 1–11, 2018.
- [10] L. Jayasinghe, T. Samarasinghe, C. Yuen, J. C. N. Low and S. S. Ge, "Temporal convolutional memory networks for remaining useful life estimation of industrial machinery," in *IEEE Int. Conf. on Industrial Technology (ICIT2019)*, Melbourne, Australia, 2018.
- [11] R. Zhao, R. Yan, J. Wang and K. Mao, "Learning to monitor machine health with convolutional bi-directional LSTM networks," *Sensors*, vol. 17, no. 2, pp. 273, 2017.
- [12] S. Bai, J. Z. Kolter and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *ArXiv Preprint*. arXiv:1803.01271, 2018.
- [13] N. Gugulothu, V. TV, P. Malhotra, L. Vig, P. Agarwal *et al.*, "Predicting remaining useful life using time series embeddings based on recurrent neural networks," *ArXiv Preprint*. arXiv:1709.01073, 2017.
- [14] W. Yu, I. Y. Kim and C. Mechefske, "Remaining useful life estimation using a bidirectional recurrent neural network based autoencoder scheme," *Mechanical Systems and Signal Processing*, vol. 129, pp. 764–780, 2019.
- [15] F. O. Heimes, "Recurrent neural networks for remaining useful life estimation," in *IEEE Int. Conf. on Prognostics and Health Management*, pp. 1–6, 2008.
- [16] A. Saxena, K. Goebel, D. Simon and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *IEEE Int. Conf. on Prognostics and Health Management*, pp. 1–9, 2008.
- [17] G. S. Babu, P. Zhao and X. L. Li, "Deep convolutional neural network-based regression approach for estimation of remaining useful life," in *Int. Conf. on Database Systems for Advanced Applications*, Springer, Cham, pp. 214–228, 2016.

- [18] F. Li, L. Zhang, B. Chen, D. Gao, Y. Cheng *et al.*, “A light gradient boosting machine for remaining useful life estimation of aircraft engines,” in *IEEE 21st Int. Conf. on Intelligent Transportation Systems (ITSC)*, pp. 3562–3567, 2018.
- [19] X. Zhang, P. Xiao, Y. Yang, Y. Cheng, B. Chen *et al.*, “Remaining useful life estimation using CNN-XGB with extended time window,” *IEEE Access*, vol. 7, pp. 154386–154397, 2019.
- [20] S. Zheng, K. Ristovski, A. Farahat and C. Gupta, “Long short-term memory network for remaining useful life estimation,” in *IEEE Int. Conf. on Prognostics and Health Management (ICPHM)*, pp. 88–95, 2017.
- [21] P. Lim, C. K. Goh and K. C. Tan, “A time window neural network-based framework for remaining useful life estimation,” in *IEEE Int. Joint Conf. on Neural Networks (IJCNN)*, pp. 1746–1753, 2016.
- [22] E. Ramasso and A. Saxena, “Performance benchmarking and analysis of prognostic methods for CMAPSS datasets,” *International Journal of Prognostics and Health Management*, vol. 5, pp. 1–15, 2014.
- [23] E. Ramasso and A. Saxena, “Review and analysis of algorithmic approaches developed for prognostics on CMAPSS dataset,” in *Annual Conf. of the Prognostics and Health Management Society*, Fort Worth, TX, USA, pp. 612–622, 2014.
- [24] H. Richter, “Engine models and simulation tools,” in *Advanced Control of Turbofan Engines*. New York: Springer, pp. 19–33, 2012.
- [25] J. B. Coble and J. W. Hines, “Prognostic algorithm categorization with PHM challenge application,” in *IEEE Int. Conf. on Prognostics and Health Management*, pp. 1–11, 2008.
- [26] Z. Zhao, B. Liang, X. Wang and W. Lu, “Remaining useful life prediction of aircraft engine based on degradation pattern learning,” *Reliability Engineering Safety*, vol. 164, pp. 74–83, 2017.