Mechanical Systems and Signal Processing 149 (2021) 107322

Contents lists available at ScienceDirect

Mechanical Systems and Signal Processing

journal homepage: www.elsevier.com/locate/ymssp

Analysis of different RNN autoencoder variants for time series classification and machine prognostics

Wennian Yu [a,*], Il Yong Kim [b], Chris Mechefske [b]

[a] College of Mechanical Engineering, Chongqing University, Chongqing 400044, PR China
[b] Department of Mechanical and Materials Engineering, Queen's University, Kingston, ON K7L 3N6, Canada

A R T I C L E   I N F O

Article history:
Received 17 June 2019
Received in revised form 24 August 2020
Accepted 23 September 2020
Available online 7 October 2020

Keywords:
Recurrent neural network
Autoencoder
Time series
Remaining useful life
Random search

A B S T R A C T

Recurrent neural network (RNN) based autoencoders, trained in an unsupervised manner, have been widely used to generate fixed-dimensional vector representations or embeddings for varying length multivariate time series. These embeddings have been demonstrated to be useful for time series reconstruction, classification, and creation of health index (HI) curves of machines being used in industrial applications, based on which the remaining useful life (RUL) of machines can be estimated. In this study, we extend the traditional form of RNN autoencoders as a feature extractor for multivariate time series to a more general form in terms of arranging the order of input or output sequences and the hidden unit architecture. We apply the embeddings obtained by different variants of RNN autoencoders for a time series classification task and a machine RUL estimation problem using two publicly available datasets. A random research strategy is used to find the optimal hyperparameters of all variants for each task in order to give a fair comparison of the general performance among different variants over a large hyperparameter space, as well as the best performance that each variant can achieve compared with the best reported values in the literature. Our results show that traditional reversing the order of output time series while maintaining the order of input time series when training an RNN autoencoder does not show improved performance for the two studied cases. Thus, intentionally arranging the input or output order seems unnecessary for training the RNN autoencoder as a feature extractor of time series. We further observe that only the RNN architectures with gating mechanism can achieve the functionality of encoding for the time series, and none of the three common gated architectures we studied shows significantly and consistently improved performance compared to the others on the two studied cases. However, the bidirectional RNN autoencoders yield slightly better performance than their unidirectional counterparts.

1. Introduction

Time series data are sampling points taken from a continuous, real-valued process over time. Many time series, e.g. multi-sensor readings recording machine degradation, contain several unique properties such as strong noise levels, high dimensionality, explicit temporal dependencies and non-stationarity, making them challenging to analyze [1]. There has been an

* Corresponding author.
E-mail addresses: wennian.yu@cqu.edu.cn (W. Yu), kimiy@queensu.ca (I.Y. Kim), chris.mechefske@queensu.ca (C. Mechefske).

https://doi.org/10.1016/j.ymssp.2020.107322
0888-3270/© 2020 Elsevier Ltd. All rights reserved.

Nomenclature

Abbreviations

CA	Classification accuracy
CWRU	Case west reserve university
CMAPSS	Commercial modular aero-propulsion system simulation
DE, FE	Drive end, Fan end
FN, FP	False negative, False positive
GRU, BiGRU	Gated recurrent neural network and its bidirectional counterpart
HI	Health index
IRF	Inner race fault
LR	Linear regression
LSTM, BiLSTM	Long short term memory, and its bidirectional counterpart
NC	Normal condition
NMT	Neural machine translation
ORF	Outer race fault
PLSTM, BiLSTM	Peephole long short term memory, and its bidirectional counterpart
RNN, BiRNN	Recurrent neural network, and its bidirectional counterpart
RNN-ED, BiRNN-ED	Recurrent neural network encoder-decoder, and its bidirectional counterpart
RUL	Remaining useful life
REF	Rolling element fault
SVM	Supporter vector machine
TN, TP	True negative, True positive

Main notations

C	the penalty parameter of SVM
e_i	the difference between the estimated RUL and the actual RUL for the *i*th test engine in the CMAPSS datasets
E	the squared reconstruction error
F	the F-score value for a classifier
H_i, $H_i'(i = 1, 2, \ldots, t)$	the hidden state series of the encoder, and the decoder respectively
$H_i'^f$, $H_i'^b(i = 1, 2, \ldots, t)$	the forward and backward hidden state series of the decoder respectively
$\boldsymbol{H_F}$	the final hidden state of BiRNN-ED
I_i, I_i' $(i = 1, 2, \ldots, t)$	the input series and output series of the RNN autoencoder
J	the number of instances
R	the step size of the sliding window along $\boldsymbol{X}$
S	the score value of a prognostics model on CMAPSS dataset
W	the length of window
$\boldsymbol{X}$	a time series
$y_i(i = 1, 2, \ldots, N)$	the predicted label for the *i*th instance of a classifier in Eq. (5)
$\boldsymbol{z_t}$	the concatenation of final hidden states from all hidden layers of the encoder

Greek Symbols

$\boldsymbol{\Omega}$, $\boldsymbol{\Omega'}$	the set of input and output time series of the RNN-EDs respectively
$\boldsymbol{\Xi}$	the set of subsequences created after sliding a window along $\boldsymbol{X}$
γ	the kernel scale of SVM
$\gamma_i(i = 1, 2, \ldots, q)$	a label space with q possible labels
$\Upsilon_i(i = 1, 2, \ldots, N)$	the ground-truth label for the *i*th instance in Eq. (5)
η	the learning rate of RNN-EDs
λ	the relaxing factor in the similarity-based curve matching technique
β	a coefficient in the similarity-based curve matching technique
W_s	the size of moving average window to smooth the HI curve in the similarity-based curve matching technique

intensive interest on the time-series research within the data mining community. After an intensive review in algorithms for time series classification, Fawaz et al. [2] found that most outperforming algorithms share one common property, which is the additional data transformation phase where the original time series are transformed into a new feature domain, based on which the subsequent data mining algorithms are performed. For instance, the dynamic time warping distance is one of the popular features that measures the similarity among time series with varying lengths. It is often coupled with a classifier such as the nearest neighbor which has been proved to be a strong baseline [3]. Time series shapelets is another noteworthy concept, which are maximally discriminative sub-sequences of time series [4]. It provides the most representative features from the original data. In the last few years, a novel unsupervised feature learning (or representation learning) for time series using recurrent neural network (RNN) based autoencoders has been proposed for time series reconstruction [5], classification [6] and machine health monitoring [7–9]. This is of great importance in industry applications considering that collecting a huge amount of labelled sensor data for training a deep learning model is usually expensive and intractable whereas collecting unlabeled sensor data is much easy and convenient. Moreover, unsupervised representation learning, such as RNN autoencoders, provides a delicate solution allowing the model to learn the internal features of the unlabeled data itself instead of being hand-crafted.

RNN autoencoder is the special case of the RNN based Encoder-Decoder (RNN-ED) model when trained to recover the input. As a variant of RNN networks, the RNN based Encoder-Decoder (RNN-ED) model, was first proposed to address sequence-to-sequence (seq2seq) learning problems [10]. It consists of a pair of RNNs, usually with the same architecture, called the encoder RNN and decoder RNN respectively. In a typical seq2seq model such as the RNN-ED model, the encoder transforms the source sequence (e.g. words, speech, video, etc.) to a fixed-dimensional vector representation, whereas the decoder maps it to the target sequence. However, it is rather difficult to train the RNN-ED due to the limitation of vanilla (standard) RNNs to learn long-range temporal dependencies. Thus, the Long Short-Term Memory (LSTM) [11] and its variants [12,13], known for dealing with this problem, are usually employed in this Encoder-Decoder setting. This encoding strategy has been successfully used in many seq2seq learning tasks, such as neural machine translation (NMT) [10,12] and image captioning [14]. It was later employed in the field of prognostics and health management to process multisensor time series data, serving as a feature extractor for time series by training the RNN-ED to recover the input in an unsupervised manner [6–9], i.e. RNN autoencoder.

Nevertheless, depending on the application and research objective, the structure of the RNN-EDs developed by different researchers are slightly different mainly in terms of the architecture of RNN hidden units (referred to also as RNN architecture in the remainder of this paper), the sequence order when training the model, and the way data is fed to the decoder RNN. For instance, Cho et al. [12] introduced a LSTM-like architecture called Gated Recurrent Unit (GRU) to build the RNN-ED (i.e. GRU-ED) for statistical machine translation, which consists of a pair of GRU based RNN with the same architecture called encoder RNN and decoder RNN respectively. The prediction (output) from the decoder at each time step (except at the first time step) is conditional upon three inputs: the final hidden state of the encoder RNN, the hidden state and the output of the decoder RNN at the previous time step. Sutskever et al. [10] proposed a multilayered LSTM based RNN-ED (i.e. LSTM-ED) for general machine translation problems. Their model differs from Cho's in two distinct ways. First, the decoder output at a given time step is determined only by two inputs: the hidden state and the output of the decoder at its direct previous time step. Second, they reversed the order of the input sequence (but not the output sequence) to improve the performance as reversing input may introduce many short term dependencies between the input and output sequences. Malhotra et al. [7,9] applied a similar LSTM-ED model on the multi-sensory data, where the model was trained to recover input time series in reverse order (i.e. reversing the output instead of the input). The reconstruction performance of the model trained by the time series collected during the machine normal (healthy) state was used for anomaly detection [9] and to obtain the health index (HI) of the machine in an unsupervised manner [7]. They [6] also introduced *Timenet* which is designed to extract features for time series from diverse datasets in an unsupervised manner. *Timenet* is actually the encoder network of the RNN-ED using the GRU architecture and trained to recover the input time series also in reverse order. One marked difference of the *Timenet* with previous RNN-ED models is that, unlike the conventional ways of feeding decoder RNN, the only input to the decoder is the final hidden state of the encoder, and the decoder at any time step is unconditional on the last generated output, i.e. unconditional decoder as termed by Srivastava et al. [14] as opposed to the traditional conditional decoder. This ensures that the final hidden state of the encoder has all the information required to recover the input sequence via the decoder RNN. This approach has been demonstrated to learn robust embeddings (features or representations) for multivariate time series, which are useful for data classification and machine health monitoring problems [6,8,15].

RNN-EDs are now applied to many sequence-to-sequence and time series modeling problems, and various versions of RNN-ED were proposed in the last few years. A criticism arises as certain schemes of RNN-EDs are not immediately apparent. For instance, LSTM and GRU have both been used as the building architecture of RNN autoencoders, however systematic comparisons of the classification and regression performance using "features" extracted from the RNN autoencoder based on these architectures have not yet been conducted in the literature. Moreover, reversing the order of the source sequence when training seq2seq models has been demonstrated to improve the model's performance [10], however, the necessity of such reversing trick for RNN autoencoders trained to extract features from time series is not directly apparent as time series data like sensor readings collected from machines inevitably exhibit some sort of stochasticity due to environmental disturbance and measurement noise, which tends to weaken temporal dependencies within sequence. This study intends to fill these gaps through exploratory comparisons in terms of performance for time series classification and machine health estimation problems (using two publicly available datasets) among various versions of RNN autoencoder with different architectures and reversing/non-reversing tricks. The most related works include studies conducted by Jozefowicz et al. [16] and Greff et al. [13], which intend to find best RNN architecture that works universally well on every dataset.

This paper is organized as follows. Section 2 provides the traditional structures of RNN autoencoders and the bidirectional versions. Several schemes including the reversed/non-reversed input/output, common RNN architectures with gating mechanisms, and the windowing strategy are briefly described. Section 3 first introduces two publicly available datasets and corresponding performance evaluation metrics, based on which the performance of different RNN-ED variants for time series classification and machine remaining useful life (RUL) estimation problems are evaluated and compared via a random search strategy. The main conclusions to this study are drawn in Section 4.

2. RNN autoencoders

In this section, we first introduce the conventional structure of RNN-ED (assumed identical with the RNN autoencoder in the remainder of this paper) and the principle of RNN autoencoder working as a feature extractor for time series. Then, we discuss several variants of RNN-ED in terms of arranging the order of input or output time series, and the architecture of the hidden unit. We also generalize two types of RNN-ED models depending on the way of training.

2.1. Traditional structure

In deep learning, an autoencoder is a type of neural network used to learn efficient code (or embeddings, features, representations which have nearly the same meaning in deep learning) from the input data in an unsupervised manner [17]. It normally consists of an encoder and a decoder. The encoder learns to compress the input data into a short code, whereas the decoder learns to un-compress the code into a set of data that closely matches the input data. As a special form of seq2seq models, the RNN-ED can also be trained in an unsupervised manner to learn fixed-dimensional representations for the high-dimensional input time series. Fig. 1(a) shows an RNN based Encoder-Decoder network proposed by Malhotra et al. [6]. Inspired by the special structure of bidirectional RNN (BiRNN) over the unidirectional RNN to capture the time dependencies within a sequence in both forward and backward manners, Yu et al. [15] proposed a bidirectional RNN-ED (BiRNN-ED) to learn more robust embeddings for the input time series, as shown in Fig. 1(b).

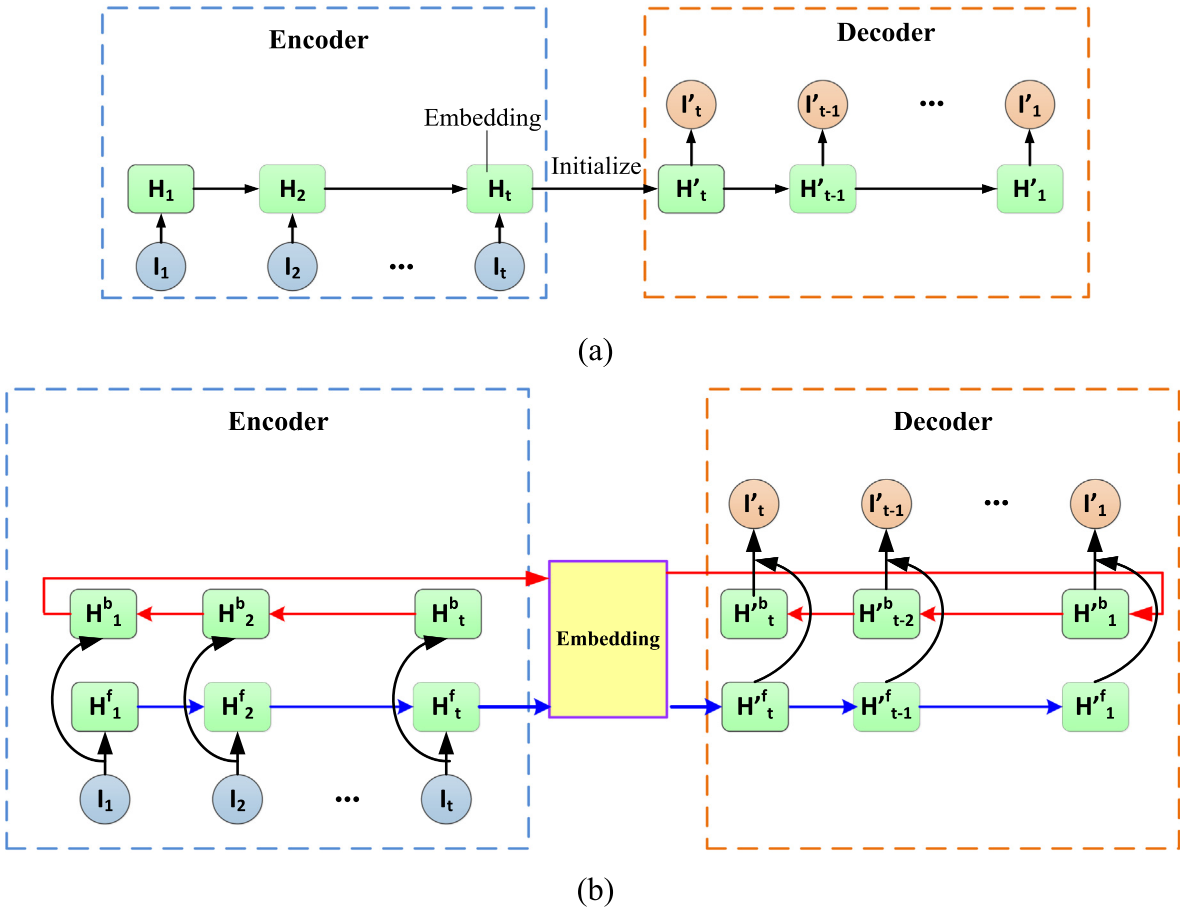


Fig. 1. Traditional structures of RNN based Encoder-Decoder: (a) unidirectional RNN-ED [6], and (b) bidirectional RNN-ED [15] (Note that only one hidden layer is shown in the graphs).

Given an input time series $\boldsymbol{\Omega} = \{\boldsymbol{I}_1, \boldsymbol{I}_2, \ldots, \boldsymbol{I}_t\}$, where t is the length of the time series and each point $\boldsymbol{I}_i$ ($i = 1, 2, \ldots, t$) is an m-dimensional vector, the encoder BiRNN iterates through each point $\boldsymbol{I}_i$ in $\boldsymbol{\Omega}$ in both forward and backward manners so that the final hidden state is obtained:

$$\boldsymbol{H}_F = \boldsymbol{H}^f{}_t \oplus \boldsymbol{H}^b{}_1 = f_e(\boldsymbol{\Omega}) \tag{1}$$

where $\boldsymbol{H}_t^f$ and $\boldsymbol{H}^b{}_1$ are the final hidden states resulting from the forward and backward processes respectively. The decoder BiRNN has the same structure as the encoder BiRNN. Unlike the seq2seq models developed in [7,10,12], which require multiple inputs to predict the decoder output at each time step, the decoder BiRNN uses only the final hidden state of encoder, namely $\boldsymbol{H_F}$ as the initial input to reconstruct the input time series $\boldsymbol{\Omega}$ in a reverse order $\boldsymbol{\Omega}' = \{\boldsymbol{I}'_t, \boldsymbol{I}'_{t-1}, \ldots, \boldsymbol{I}'_1\}$ [6,15]:

$$\boldsymbol{\Omega} = f_d(\boldsymbol{H}_F) \tag{2}$$

The reconstruction error at the i^{th} time step is $\boldsymbol{e}_i = \boldsymbol{I}'_i - \boldsymbol{I}_i$. The BiRNN Encoder-Decoder is trained to minimize the squared reconstruction error E given by:

$$E = \frac{1}{2}\sum_{i=1}^{t}(\|\boldsymbol{e}_i\|_2)^2 \tag{3}$$

where $\|\cdot\|_2$ is the 2-norm operator of a vector. Once the BiRNN-ED is well trained, the final hidden state of the encoder $\boldsymbol{H}_F$ carries all the relevant information to reconstruct the input time series via the decoder. If there are multiple hidden layers of the BiRNN-ED, the embedding $\boldsymbol{z_t}$ is often obtained by the concatenation of the final hidden states from all the hidden layers of the encoder:

$$\boldsymbol{z}_t = \boldsymbol{H}_F^1 \oplus \boldsymbol{H}_F^2 \oplus \cdots \boldsymbol{H}_F^L \tag{4}$$

where $\boldsymbol{H}_F{}^l$ is the final hidden state vector at the lth layer of encoder ($l = 1, 2, \ldots L$), and L is the number of hidden layers. It is worth noting that RNN-ED (including BiRNN-ED in the remainder of this paper if not otherwise stated) is normally trained on a set of time series $\{\boldsymbol{\Omega}^{(1)}, \boldsymbol{\Omega}^{(2)}, \ldots, \boldsymbol{\Omega}^{(J)}\}$, and each time series $\boldsymbol{\Omega}^{(j)} = \{\boldsymbol{I}_1^{(j)}, \boldsymbol{I}_2^{(j)}, \ldots, \boldsymbol{I}_{tj}{}^{(j)}\}$ ($j = 1, 2, \ldots J$) may have different length t_j. Once well trained, the encoder serves as a feature extractor that can map varying length multivariate time series into fixed dimensional vector representations, on which subsequent machine learning models can be applied.

2.2. Several points with RNN-ED

2.2.1. Reversed or Non-reversed input or output

The idea of RNN-ED was first proposed by Cho et al. [12] for seq2seq learning problems (neural machine translation), which is to map the input sequence to a fixed-size vector using an encoder RNN, and then map the vector to the target sequence using an decoder RNN (we refer to it as Cho's NMT model). Sutskever et al. [10] found that reversing the order of the input sequence improve the performance of RNN-ED markedly as dosing so may introduce many short-term dependencies between the input sequence and output sequence which makes the optimization easier (we refer to it as Sutskever's NMT model). Motivated by this reversing trick, Srivastava et al. [14] proposed a RNN based autoencoder, of which the decoder is trained to recover the input sequence in reverse order, as shown Fig. 1 (we refer to it as Srivastava's NMT model). Inspired by their work, we concludes 4 modes of RNN-ED based on whether the order of input or output is reversed or not, as listed in Table 1. They are: Mode #1, non-reversed input and non-reversed output (NINO); Mode #2, non-reversed input and reversed output (NIRO); Mode #3, reversed input and non-reversed output (RINO); and Mode #4, reversed input and reversed output (RIRO). Thus, Cho's NMT model, Srivastava's NMT model and Sutskever's NMT model belong to Mode #1, #2 and #3 respectively, though they are first designed for processing sequences (e.g. texts) instead of time series (e.g. sensor readings). Compared with Mode #1 and Mode #4, the orders of input and output are opposite to each other in Mode #2 and Mode #3 as doing so may improve the performance for seq2seq problems. Inspired by this reversing strategy, Malhotra et al. first applied the RNN autoencoder to extract features from time series based on Srivastava's NMT model, which belongs to Mode #2 as described in Section 2.1. However, time series data like sensor readings is different from sequence data like texts as the logical dependencies within sensor readings are not obvious. Thus, for the RNN-ED trained on the time series, the necessity of such reversing trick is not directly apparent. Its efficacy warrants further investigation, which constitutes one of the major objective of this study.

Table 1
The four modes of the RNN-ED (Note $\boldsymbol{I}'_t$ is the reconstructed $\boldsymbol{I}_t$ from the decoder output).

Mode	Name	Input time series	Output time series
Mode #1	NINO	$\{\boldsymbol{I}_1, \boldsymbol{I}_2, \ldots, \boldsymbol{I}_t\}$	$\{\boldsymbol{I}'_1, \boldsymbol{I}'_2, \ldots, \boldsymbol{I}'_t\}$
Mode #2	NIRO	$\{\boldsymbol{I}_1, \boldsymbol{I}_2, \ldots, \boldsymbol{I}_t\}$	$\{\boldsymbol{I}'_t, \boldsymbol{I}'_{t-1}, \ldots, \boldsymbol{I}'_1\}$
Mode #3	RINO	$\{\boldsymbol{I}_t, \boldsymbol{I}_{t-1}, \ldots, \boldsymbol{I}_1\}$	$\{\boldsymbol{I}'_1, \boldsymbol{I}'_2, \ldots, \boldsymbol{I}'_t\}$
Mode #4	RIRO	$\{\boldsymbol{I}_t, \boldsymbol{I}_{t-1}, \ldots, \boldsymbol{I}_1\}$	$\{\boldsymbol{I}'_t, \boldsymbol{I}'_{t-1}, \ldots, \boldsymbol{I}'_1\}$

Fig. 2 shows the structure of BiRNN-ED in Mode #1 used to reconstruct the input time series in a non-reversed order, i.e. $\boldsymbol{\Omega}' = \{\boldsymbol{I}'_1, \boldsymbol{I}'_2, \ldots, \boldsymbol{I}'_t\}$ in Eq. (2). In the following study, we will investigate the robustness of the embeddings obtained from the abovementioned 4 modes for time series classification and machine health monitoring tasks. The general performance among the 4 variants of RNN-ED on two specific datasets will be evaluated and compared.

2.2.2. Architecture of hidden units in RNN-ED

The architecture of RNN hidden units directly determines how the RNN processes the information passing through it. Vanilla (standard) RNNs like the Elman network [18] and the Jordan network [19] suffer from severe vanishing and exploding gradient problems when training them using the backpropagation through time algorithm, especially on sequences with long-range temporal dependencies. In fact, Sutskever et al. [10] found that they were unable to train a standard RNN autoencoder on the general non-reversed translational problem. Nevertheless, they guessed that it may be trainable when the source sequences are reversed (but with no experimental validations). Yu et al. [15] found that it is difficult to train a standard RNN autoencoder to reconstruct the reversed input time series. Thus, we conjecture that only the RNN architectures with gating mechanism to deal with the vanish gradient problem can achieve the functionality of encoding for the input time series as this gating mechanism enables the RNN to learn long range temporal dependencies within the time series.

In this study, we will investigate three popular types of building units for hidden layers of RNN autoencoder: the standard long short-term memory unit (LSTM) [11], the peephole long short-term memory unit (PLSTM) [13], and the gated recurrent unit (GRU) [12]. They all adopt the gating mechanism, and the architectures of these units are shown in Table 2. The standard LSTM unit has three gates (i.e. input, forget and output gates) and a cell to control the information passing through it. The GRU, on the other hand, uses only two gates (i.e. reset and update gate whose functions are similar to the input and forget gates of LSTM), and omits the cell and the output gate, meaning that unlike the LSTM unit, the GRU exposes its full content without any control. For both the LSTM unit and GRU, the state of gates are determined by the input of current state I_t and hidden state of previous time step H_{t-1}, whereas for the peephole LSMT unit, there is an external peephole connection from the cell to the gates in order to "make precise timings easier to learn" as argued by the invertors Gers and Schmidhuber [20]. Obviously, training an RNN composed of the PLSTM unit consumes the most computation time among the three units, seconded by the standard LSTM unit, and then the GRU. However, it doesn't mean that a complex internal architecture should perform better than a simple one. Greff et al. [13], Chung et al. [21], and Jozefowicz et al. [16] all have done comparison studies among LSTM variants on a large number of tasks, and could not make concrete conclusion on which types of gating mechanism would perform better in the general case. Interestingly, Jozefowicz et al. found that adding the bias of 1 to the forget gate of the LSTM unit significantly improves its performance on tasks where it falls behind the GRU. Thus, they recommend setting the bias to 1 for the forget gate of the LSTM unit in every application, which is adopted in this study.

2.2.3. Static models and dynamic models

Typically, if the length of time series is large, it is a usual practice to partition the time series into windowed time series (subsequences) with fixed length [7,8,15]. Considering a window with a fixed size W sliding along the time series $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\}$ with a step size (stride) R, there will be $N = (T\text{-}W)/R + 1$ subsequences (padding the last vector values of time series in the last window if the data in the last window does not fit the window size) denoted by $\boldsymbol{\Xi} = \{\boldsymbol{\Omega}_1, \boldsymbol{\Omega}_2, \ldots \boldsymbol{\Omega}_N\}$, and for the ith window, $\boldsymbol{\Omega}_i = \{\boldsymbol{x}_{1+(i-1)*S}, \boldsymbol{x}_{2+(i-1)*S}, \ldots, \boldsymbol{x}_{W+(i-1)*S}\}$. It should be noted that the stride R should not be larger than the window size W so that each point in the original time series $\boldsymbol{X}$ can be included in the resulting windowed time series $\boldsymbol{\Xi}$. This is just for one instance of time series data. Considering a population of time series $\boldsymbol{X}^{(j)} = \{\boldsymbol{x}_1^{(j)}, \boldsymbol{x}_2^{(j)}, \ldots, \boldsymbol{x}_{Tj}^{(j)}\}$ (j = 1, 2, ... J where J is the total number of instances), and these instances may have different time length T_j. Similarly, we can create subsequences for each instance using the same sliding window size W and stride R. The resulting subsequences for all instances have the same

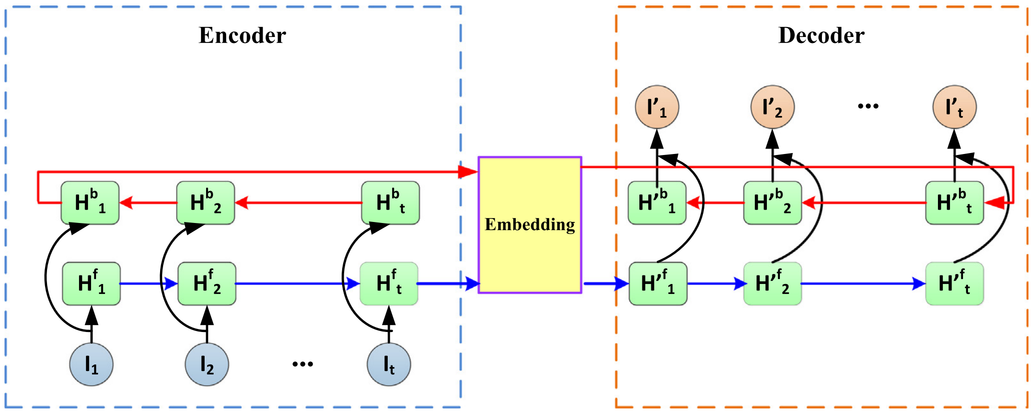
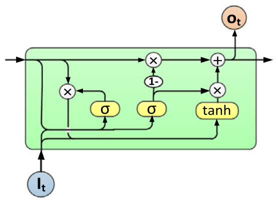
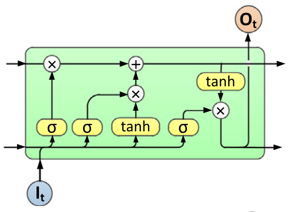
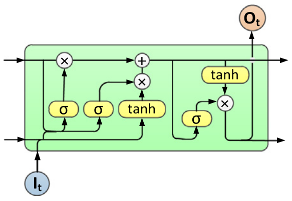


Fig. 2. The mode #1 of BiRNN-ED (Note its difference with respect to Fig. 1(b)).

Table 2
The popular architectures of the RNN hidden units with gating mechanism.

	Structure	Notations
GRU [12]		⊗ element-wise product ⊕ sum 1- subtraction from 1 σ sigmoid function tanh hyperbolic tangent
LSTM [11]		⊗ element-wise product ⊕ sum σ sigmoid function tanh hyperbolic tangent
PLSTM [13]		⊗ element-wise product ⊕ sum σ sigmoid function tanh hyperbolic tangent

Note: $\boldsymbol{I}_t$ and $\boldsymbol{O}_t$ are the input and output at the tth step. The details regarding the mathematic operations behind these three architectures can be found in [11–13].

length, i.e. W, though the original time series for these instances may have varying time length. The abovementioned RNN-ED models can be trained on these windowed time series. A fixed-dimensional representation or embedding was thus obtained for each subsequence. RNN-ED models trained on these fixed-size windowed data are what we call static models, or models trained in the static manner. On the other hand, RNN-ED models can also be trained directly on the diverse time series with varying length without the need of windowing, as done in [6]. If trained in this way, we refer to the RNN-ED models as dynamic models, or models trained in the dynamic manner, since the length of model input is varying dependent on the length of input time series.

3. Results and discussions

In this section, we apply the embeddings obtained by different variants of RNN autoencoder for a time series classification task and a machine RUL estimation problem using two publicly available datasets. Comprehensive comparisons are made among different variants of RNN autoencoder based on the classification and prognostic performance which are expressed by boxplots.

3.1. Datasets and task descriptions

3.1.1. CWRU bearing dataset and performance evaluation metrics

The bearing data provided by the CWRU (Case Western Reserve University) was collected from a test rig consisting of an electric motor driving a shaft which is loaded by a dynamometer and an electronic control system [22]. The test bearings include the drive end (DE) bearing and the fan end (FE) bearing supporting the motor shaft. Various sizes of single point faults were introduced to the inner race, a rolling element and the outer race of test bearings respectively. The data contains acceleration signals collected in the vertical directions on the DE and on the FE of the motor housing during the bearing states of normal condition (NC), inner race fault (IRF), rolling element fault (REF) and the outer race fault (ORF). It should be pointed out that the ORFs are further grouped into three cases depending on the fault location relative to the load zone (i.e. ‘centred’, ‘orthogonal’ and ‘opposite’). The motor was run nearly at a constant speed under four different torques (0, 1, 2 and 3 hps) applied by the dynamometer. However it was pointed out in [23] that the load here is meaningless as there is no gears to transform the torque to a radial load borne by the test bearings. Detailed descriptions of the CWRU bearing datasets can be found in [23]. In this study, we create a four-class time series classification task. To do so, we used a similar data pre-processing strategy as used in [24] to preprocess the dataset. The details can be referred to in Section 3.2.1.

For a multi-label classification task, it is usual to define instance-based metrics to evaluate the performance of a classifier on each instance separately and then obtain the mean value across all instances that need to be classified. For example, the classification accuracy of a classifier is defined as the percentage of correctly classified instances, i.e.:

$$CA = \frac{100}{N}\sum_{i=1}^{N} \mathrm{Cor}(y_i, \Upsilon_i) \tag{5}$$

where Cor(y_i, Υ_i) = 1 if y_i = Υ_i, otherwise Cor(y_i, Υ_i) = 0 (correct classification); Υ_i is the ground-truth label for the *i*th instance, and $\Upsilon_i \in \{\gamma_1, \gamma_{2,} \ldots, \gamma_q\}$, which is the label space with q possible labels (which is 4 in this case); y_i is the predicted label for the *i*th instance, and $y_i \in\{\gamma_1, \gamma_{2,} \ldots, \gamma_q\}$; N is the total number of instances. However, this metric tends to be overly strict especially when the size of label space (i.e. q) is large [25].

It is also common to define label-based metrics to evaluate the performance of a classifier on each class label separately and then obtain the mean value across all labels. For example, the precision (P), recall (R) and F-score (F) of a classifier on a label γ_j are defined as [25] :

$$P(\gamma_j) = \frac{TP_j}{TP_j + FP_j} \tag{6}$$

$$R(\gamma_j) = \frac{TP_j}{TP_j + FN_j} \tag{7}$$

$$F(\gamma_j) = \left(\frac{P(\gamma_j)^{-1} + R(\gamma_j)^{-1}}{2}\right)^{-1} \tag{8}$$

where TP_j, FP_j, TN_j and FN_j represent the number of *true positive*, *false positive*, *true negative* and *false negative* instances with respect to the label γ_j, i.e.:

$$TP_j = \sum_{i=1}^{N} \mathrm{Sat}(y_i = \gamma_j \ \text{ and } \ \Upsilon_i = \gamma_j) \tag{9}$$

$$FP_j = \sum_{i=1}^{N} \mathrm{Sat}(y_i \neq \gamma_j \ \text{ and } \ \Upsilon_i = \gamma_j) \tag{10}$$

$$TN_j = \sum_{i=1}^{N} \mathrm{Sat}(y_i \neq \gamma_j \ \text{ and } \ \Upsilon_i \neq \gamma_j) \tag{11}$$

$$FN_j = \sum_{i=1}^{N} \mathrm{Sat}(y_i = \gamma_j \ \text{ and } \ \Upsilon_i \neq \gamma_j) \tag{12}$$

where Sat(*argument*) = 1 if the *argument* is true, otherwise Sat(*argument*) = 0. In this paper, we will use the macro-averaging F-score metric to evaluate the classification performance of classifiers, which is the mean value of F(γ_j) across all labels, i.e. [25]:

$$F = \frac{1}{q}\sum_{j=1}^{q} F(\gamma_j) \tag{13}$$

The larger the F-score value, the better the classifier's performance.

3.1.2. CMAPSS engine dataset and performance evaluation metrics

The CMAPSS dataset contains time series of 21 sensor readings for the complete run-to-failure degradation of 100 turbofan engines (i.e. *train_FD001.txt*) [26], which was created using the turbofan engine simulation model called C-MAPSS (Commercial Modular Aero-Propulsion System Simulation) [27]. A similar dataset was provided for 100 test engines with incomplete data that end prior to failure (i.e. *test_FD001.txt*), whose remaining useful live (RUL) need to be determined by the users' prediction models that should be trained on the corresponding training dataset. The actual RULs of the test engines were given in *RUL_FD001.txt*, based on which users can evaluate the performance of their models on their own. A *Score* metric is defined by the data creator to quantitively assess the prediction performance of RUL prediction models proposed by different users [27]. It was defined as:

$$S = \begin{cases} \sum_{i=1}^{N} \left(\exp\left(-\frac{e_i}{13}\right) - 1\right), \textit{if} \;\; e_i \le 0 \\ \sum_{i=1}^{N} \left(\exp\left(\frac{e_i}{10}\right) - 1\right), \text{if} \;\; e_i > 0 \end{cases} \tag{14}$$

where N is the number of test engines (which is 100 in this case), and e_i is the difference between the estimated RUL and the actual RUL for the ith test engine (e_i = estimated RUL – actual RUL). The smaller the value of the score S, the better the performance of the prediction model.

Another commonly-used metric is accuracy, which evaluates the percentage of correct predictions among all instances [7,8,15]. A correct prediction is confirmed when the prediction error e_i falls within the range of -13 and 10, i.e. $e_i \in [-13, 10]$. Thus,

$$PA = \frac{100}{N} \sum_{i=1}^{N} \text{Cor}(e_i) \tag{15}$$

where $\text{Cor}(e_i) = 1$ if $e_i \in [-13, 10]$, otherwise $\text{Cor}(e_i) = 0$ (incorrect prediction). A prediction is considered as false positive (FP) if $e_i < -13$, and false negative (FN) if $e_i > 10$.

In the literature, this dataset was mostly treated as a regressive prediction task for machine health assessment, though a few used it for anomaly detection problem (seen as a 1-class classification) or multi-class classification task for differentiations among diffident machine degradation levels [28].

3.2. Time series classification

3.2.1. Data preparation and supporter vector machine

In this section, we demonstrate the performance of using embeddings obtained by various RNN-ED models for the time-series classification task from the experimental data taken by the bearing data center in the CWRU (Case Western Reserve University). We use only the smallest size (0.007 in.) of the *48 k Drive End Bearing faults Data* and the *Normal Baseline Data* under the 3hps for incipient fault classification analysis, i.e. IRF, REF, ORF and NC. For the ORF, we select the data corresponding to the fault at the 'centred' location. For each bearing health condition, the data includes vibration signals acquired from both the FE and the DE sampled at a 48 kHz with a duration of 5 s. In order to create a time series classification task from these data, for each vibration signal (with a length of 240 K), we partition the signal into 80 non-overlapped subsequences. Thus, for each bearing health condition, there are 80 time-series samples, and each sample includes two-dimensional sensor readings whose length is fixed to 3 K. We then adopt the local-feature extraction strategy [24] with a window length of 100 so at to make the feature samples much shorter. The features we select are all time-domain features including {rms, mean, variance, peak, kurtorsis, skewness}. As a whole, there are 320 labeled samples and each sample has a dimension of 12. We randomly select 80% as the training set, and the remaining 20% are used to test the performance of the classification performance of different models.

We first use the z-score normalization to transform the time series within an acceptable range before inputting them into the RNN-ED models, and train the models on these samples in the dynamic manner as described in Section 2.2.3. The encoder of the trained model can thus be used to transform the original time series into fixed-dimensional embeddings. Due to the scarcity of the data points, dropout is used to regularize the RNN-ED models and is applied only to the non-recurrent connections [6]. We use the multi-class nonlinear supporter vector machine (SVM) as the classifier and train it on the embeddings obtained from the RNN-ED models. The optimization process of SVM is based on structural risk, thus it has a better generalization capacity and strong resistance to the over-fitting problem than most other approaches. In this study, the commonly-used radial basis function (RBF) is employed as the kernel function of the SVM due to its several good merits over other kernel functions (fewer hyperparameters and numerical difficulties) [29]. The optimal values of two associated hyperparameters, i.e. penalty parameter C and kernel scale γ, are chosen from a bounded parameter space via the Bayesian optimization that minimizes the 5-fold cross-validation error on the training set of the CWRU bearing dataset we recreated. The trained SVM with optimal parameter settings is thus used to classify the testing set.

3.2.2. Hyperparameter search

One major difficulty of comparing performance among various models is that different models may require different settings of hyperparameters to give good performance [13], and we are interested in the comparisons of the general performance of different models over a large hyperparameter space and the best performance that the model can achieve. In order to provide fair comparisons, we propose to use the random search method to scan over the approximate same hyperparameter space for each model and obtain their individual good-performing hypermeters' settings. Compared with the common grid search and manual search methods, random search is easy to implement and more efficient for hyperparameter optimization, especially for large parameter space [30]. It should be noted that random search tries to find the well-performing hyperparameter settings in a limited number of trials. Thus, it's important to set appropriate search range and sampling method for each hyperparameter. In this paper, the search range for each hyperparameter is set according to its property (discrete or continuous), prior domain experience through trial and error (rough locations of well-performing points) and similar strategies used in the literature [13,30]. There are some general rules that can be follow: 1) some hyperparameters (e.g. learning rate) play the major roles in the optimization as concluded by previous work, a relatively wide search range is required in order to cover more well-performing points; 2) some hyperparameters (e.g. dropout ratio) have minor effects, thus a narrow search range is sensible; and 3) for most hyperparameters, too small or too large values tend to significantly deteriorate the final performance. Thus, we have carefully set their search ranges to ensure they are well-performing. As to the sampling strategy, the following sampling rules are adopted to ensure the search efficiency: 1) for a search range spanning more than one order of magnitude, the log-uniform sampling is more efficient than the uniform sampling; 2) for a search range spanning less than one order of magnitude, the uniform sampling is preferred; and 3) for a search range with only a few discrete points, it is sensible to scan all these points with equal probability.

Based on above-mentioned rules, we perform 4 random searches for each mode listed in Table 1. Each random search includes 200 trials of random sampling of the following five hyperparameters within given search domains.

- Sensor set Ω: {sensor readings from FE}, {sensor readings from DE} and {sensor readings from both DE and FE}.
- Architecture of hidden units: GRU, LSTM, PLSTM, BiGRU, BiLSTM and BiPLSTM. It should be noted that the initial bias value of forget gate for every LSTM cell was set to 1 as suggested in [16];
- Learning rate η: float value sampled log-uniformly from 0.005 to 0.05;
- Number of hidden units: integer value sampled log-uniformly from 10 to 100 for unidirectional RNN-EDs, and 5–50 for bidirectional RNN-EDs. Note the search range is different since the number of hidden layers for bidirectional networks is two time of that for their unidirectional counterparts, and doing so makes the number of parameters between them roughly comparable [31];
- Dropout ratio: 0, 0.2, 0.5, 0.8 with equal probability.

In all cases, the number of hidden layers is fixed to 1 because it was found that multiple hidden layers does not show clear improvement in performance (possibly due to the degradation problem of deep network [32]) but the computation cost is extremely high. We have tried to choose reasonable searching range for each hyperparameter and included 200 hundred trials for each random search, which takes us only a couple of days to finish one search on a personal computer equipped with Inter Core i7 processor, 3.6 GHz clock speed and 128 GB RAM. Though not completely comprehensive, we conjecture that such a search strategy ensures us to reach sensible and general conclusions of the classification performance among various models. Similar strategies can also be found in [13,30].

3.2.3. Results and discussions

A summary of the random search results for all 4 modes of RNN-ED listed in Table 1 is shown in Fig. 3 in terms of the classification accuracy and F-score value. Fig. 3(a) shows the performance distribution over the whole 200 trials, whereas Fig. 3(b) shows the top 20% performing trials for each metric and each mode. It is worthy noticing that the notch in each box provides the 95% confidence interval of the median value (the horizontal red line), meaning if notches of two given boxes do not overlap, it can be regarded that their true median values are different with 95% confidence. For the purpose of comparison, in the application of language translation, Sutskever et al. [10] found that reversing the order of words in the source sentence but not the target sentence improved the translation performance of long sentences markedly as it enables similar meanings in the source sentence and target sentence are close to each other, making it easy to "establish communication"

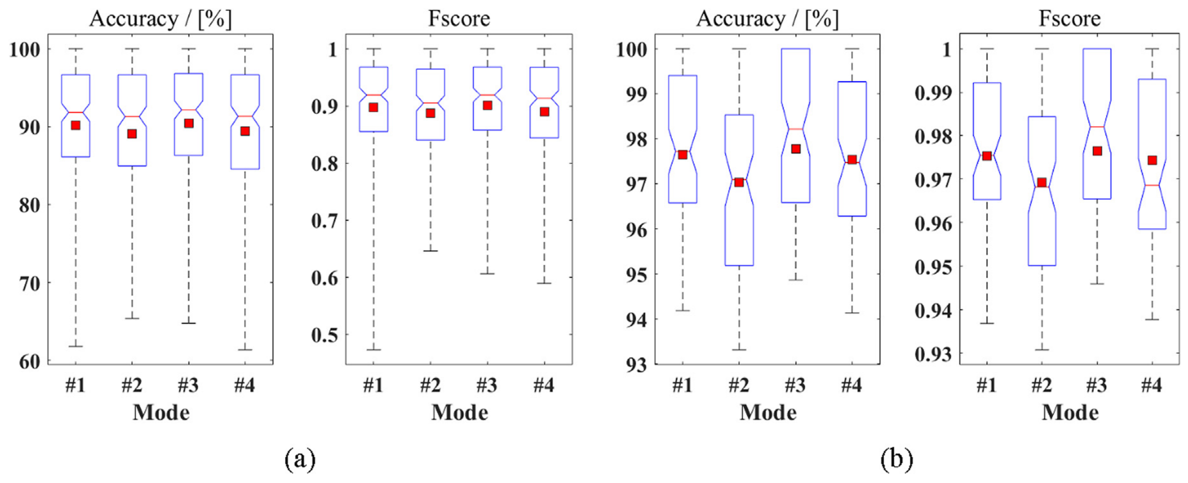


Fig. 3. The performance of the 4 RNN-ED variants listed in Table 1 via random search strategy in terms of the classification accuracy and F-score value: (a) for all 200 trials, and (b) for the best 20% trials (Note in the boxplot, the box represents the range between the 25th and 75th percentile of the data; the whiskers indicate the whole range of the data; the horizontal red line indicates the median value whereas the red dot indicates the mean value; the notch provides the 95% confidence interval of the median value). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

between the input and the output. However, in our case where the input and output are such sequences as time series, it can be found that these four modes of RNN-ED yield similar classification performance on the CWRU bearing dataset, and it's hard to tell which mode performs better than the others. This demonstrates that traditional reversing the order of the output time series while maintaining the order of the input time series (i.e. mode #2) when training an RNN autoencoder is not a necessary strategy, at least on the studied CWRU bearing dataset.

Table 3 list the classification results achieved by various methods as reported in [24], and the searching results of the best 20% trials according to the 200 trials of random search for each RNN-ED mode proposed in this work. The comparison made is not entirely fare (due to different data prepressing and training/test splitting strategies), but it sheds light on the robustness of embeddings obtained by RNN-ED variants for time series classification tasks.

In addition to the 4 random searches for each mode listed in Table 1, we also perform 6 random searches for each architecture listed in Table 2 and their bidirectional versions over 200 trials using the similar searching hyperparameters and searching ranges (the only difference is that the 4 RNN-ED modes replace the original 6 RNN-ED architectures during random sampling). Fig. 4 shows the search results for top 20% trials of each metric and each architecture. It was found that the LSTM network seems to perform worse than its two variants for both the classification accuracy and F-score metrics, which probably illustrates that the LSTM architecture does not work well on the CWRU dataset. Moreover, the bidirectional RNN architectures works slightly better than the unidirectional counterparts (especially the BiGRU which has the best performance among the all). This is because the bidirectional RNN can learn both the forward and backward dependencies within time series, and thus improve the performance, as also confirmed in [15,31].

3.3. Engine health estimation

3.3.1. Data preparation and methodology for engine RUL estimations

In this section, we demonstrate the performance of using embeddings obtained by various RNN-ED models for the RUL estimations of CMAPSS test engines. There is a total of 100 training instances (engines). Each was run for a variable number of cycles from a healthy state until failure. The length of instances varies from a minimum of 128 cycles to a maximum of 362 cycles. There is a total of 20, 631 cycles. The z-score normalization is also applied on the data before inputting them into the RNN-ED models. Since this is a regressive prediction task, we use the sliding window strategy to decompose the original time series into subsequences (i.e. the static manner described in Section 2.2.3) and train the RNN-ED models on these subsequences. One difference from [7,8,15] is that we create subsequences with varying step size R instead of fixing it to 1. Dropout is omitted in the training as the samples in this case are sufficient.

The flowchart of the whole procedure for the engine RUL estimations is described in Fig. 5 [15]. It mainly consists of two steps. The RNN-EDs are first employed to map the original run-to-failure multi-sensor readings of training engines into one-dimensional health index (HI) curves via embeddings. These HI curves represent the various degradation patterns of engines and will be maintained in a library in the off-line stage. In the online stage, the similarity-based HI curve matching technique [33] was adopted to match the test HI curve with each training HI curve in the library, from which the RUL of the test engine was estimated based on the top few training HI curves which have the most similar degradation trend with the test HI curve. The details of the methodology can be found in [8,15].

3.3.2. Hyperparameter search

Based on sampling rules described in Section 3.2.2, we also perform 4 random searches for each mode listed in Table 1. Each random search includes 200 trials by randomly sampling the following 9 hyperparameters within given search domains:

- Sensor set Ω: the four sensor subsets listed in Table 4, with the number of sensors being 3, 7, 10 and 14 respectively. Note that there are 21 sensor readings for the CMASS engines. Not all of them are informative. We select the 4 sensor subsets that have been commonly used in the literature for searching;
- Sliding window size W: integer value sampled uniformly from 2 to 40;
- Sliding window stride R: integer value sampled uniformly from 1-W. Note the stride R should not be larger than the window size W;

Table 3
Comparisons of the search results on the CWRU bearing dataset in terms of classification accuracy as reported in [24] and each mode of this work. (Note classification accuracies obtained by the proposed modes are expressed by the median and the best value of the top 20% trials).

Researchers	Approaches and Accuracies							
Zhao e al.[24]	RNN		SVM		GRU		LFGRU	
Accuracy (%)	95.6		97.2		98.1		99.6	
This work	Mode #1		Mode #2		Mode #3		Mode #4	
Accuracy (%)	Median	Best	Median	Best	Median	Best	Median	Best
	97.7	100	97.0	100	97.8	100	97.6	100

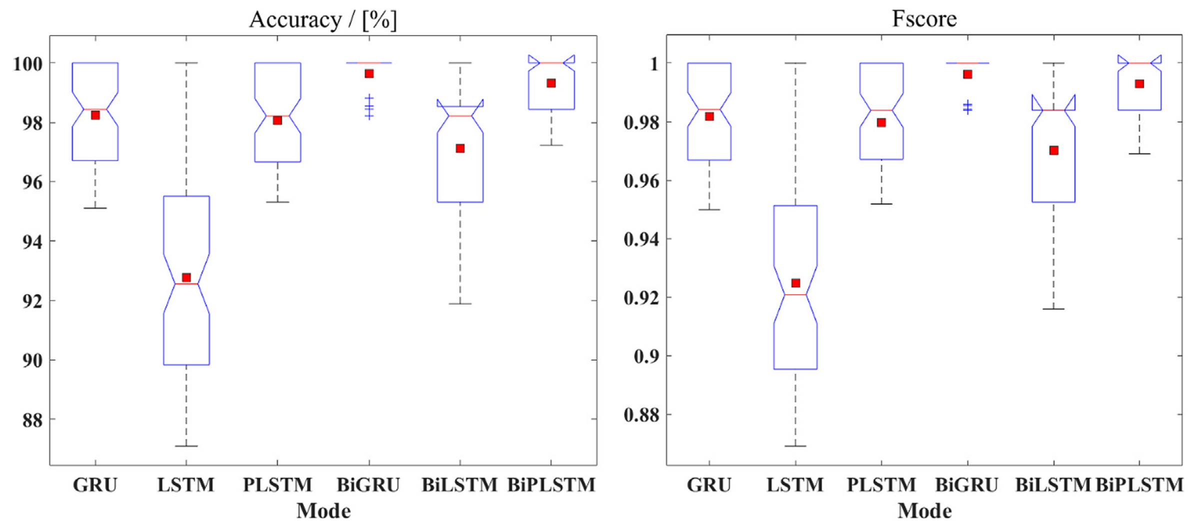


Fig. 4. The performance of the 6 RNN-ED architectures via random search in terms of the classification accuracy and F-score value for the best 20% trials.

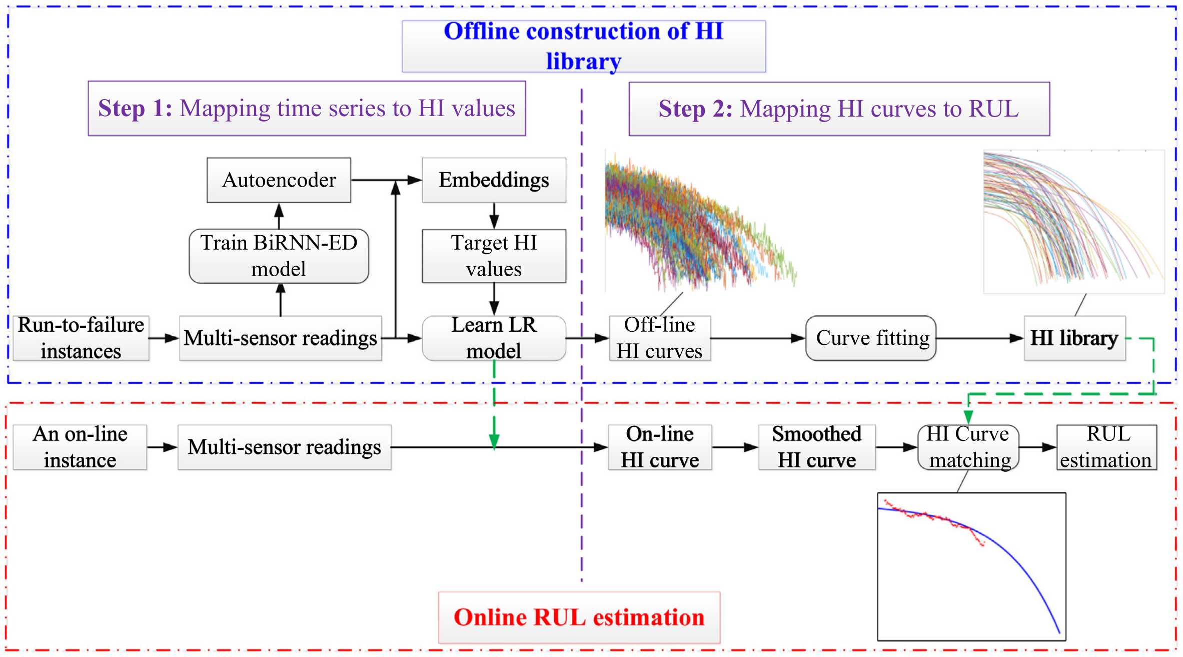


Fig. 5. Flowchart of the methodology for the engine RUL estimation [15].

Table 4
The selected 4 sensor subsets for random search.

Number of sensors	Researchers	Sensor subsets
3	Wang et al. [33]	{7, 12, 15}
7	Wang et al. [33]	{2, 3, 4, 7, 11, 12, 15}
10	Coble and Hines [34]	{2, 3, 4, 9, 11, 14, 15, 17, 20, 21}
14	Li et al. [35]	{2, 3, 4, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20 ,21}

- Architecture of hidden units: GRU, LSTM, PLSTM, BiGRU, BiLSTM and BiPLSTM. The initial bias value of forget gate for every LSTM cell was set to 1 as suggested in [16];
- Learning rate η: float value sampled log-uniformly from 0.005 to 0.05;

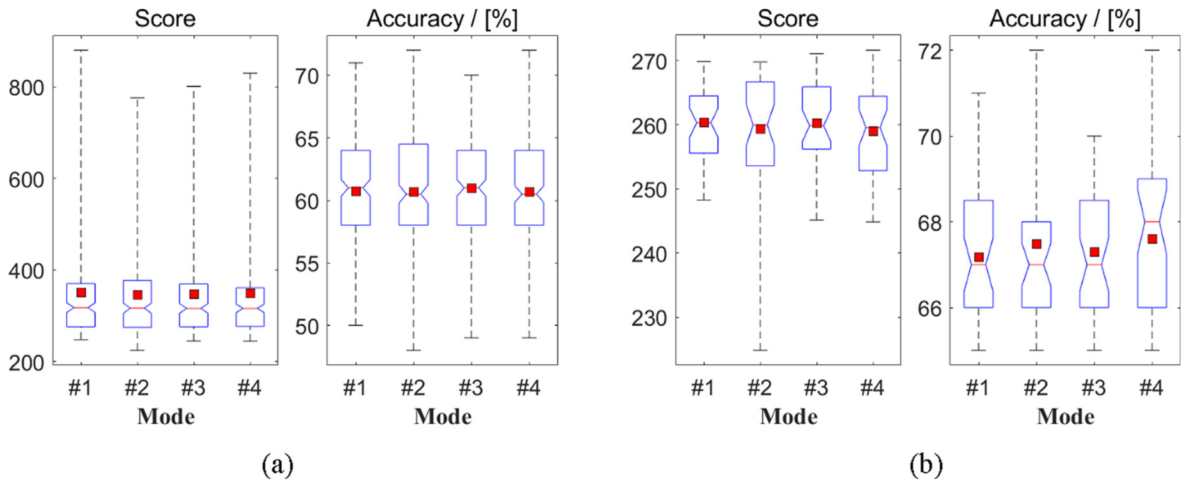


Fig. 6. The test engine RUL estimation performance of the 4 RNN-ED models listed in Table 1 for 200 trials of random search in terms of the score value and accuracy: (a) for all 200 trials, and (b) for the best 20% trials.

- Number of hidden units: integer value sampled log-uniformly from 10 to 100 for unidirectional RNN-EDs, and 5–50 for bidirectional RNN-EDs;
- Relaxing factor λ in the similarity-based curve matching technique [7,8,15]: float value sampled uniformly from 0.001 to 0.002;
- Coefficient β in the similarity-based curve matching technique [8,15]: float value sampled uniformly from 0.8 to 0.98;
- Moving average window size W_s to smooth the HI curve in the similarity-based curve matching technique [8,15]: integer value sampled uniformly from 5 to 20.

3.3.3. Results and discussions

A summary of the random search results for all 4 modes of RNN-ED is shown in Fig. 6 in terms of the score value and accuracy. Fig. 6(a) shows the performance distribution over the whole 200 trials, whereas Fig. 6(b) shows the top 20% performing trials for each metric and each mode. Same with the time classification task on the CWRU bearing dataset, it can be found that these 4 modes of RNN-ED also yield similar RUL estimation performance on the CMAPSS test engine dataset (*text_FD001.txt*), and there is no mode that certainly outperforms the others, meaning it doesn't matter how we intentionally reverse the order of input or/and output time series. At a closer look, mode #4 seems to have slightly better performance than the other three modes, and the lowest score value is achieved by mode #2. But it's hard to make concrete conclusions by these subtle nuances.

Table 6 listed the best results on the CMAPSS test engines in terms of the score value and accuracy including some reported values in the literature [7,8,36–40] and values obtained by each mode in this work. These prognostic approaches can be classified as deep learning based methods and non-deep learning based methods. It can be found from Table 5 that the prediction results using non-deep-learning methods are generally inferior than those using deep learning methods, which indicates the superiority of using deep learning tools for time series data mining. Besides, our methods based on the embeddings obtained by the RNN-ED variants are quite competitive and promising compared with state-of-art values.

Similarly, we perform another 6 random searches for each architecture of RNN-ED over 200 trials. Fig. 7 shows the results for the top 20% trials of each metric and each architecture. It can be noted that these 6 architectures yield close performance in terms of the score value, and it's hard to tell which architecture performs even slightly better than the others. As to the comparisons between the bidirectional and unidirectional RNN architectures, it seems that the former yield slightly lower score value than the latter (except the GRU), however, the accuracy does not improve for the bidirectional RNN architectures, as also noticed in [15]. The comparisons among different architectures seem inconclusive. However, one point can be made clear, that there is no architecture significantly improves over the standard LSTM in the two studied case. As also concluded by Jozefowicz et al. [16] after evaluating a variety of RNN architectures, "*if there are architectures that are much better than the LSTM, then they are not trivial to find.*"

4. Conclusions

In this study, we extend the traditional form of RNN base Encoder-Decoder (RNN-ED) as a feature extractor for multivariate time series by generalizing 4 modes and 6 architectures of RNN-ED. Two ways of training the RNN-ED models on a population of varying length time series are discussed. We evaluate the performance including time series classification and machine prognostic (i.e. RUL estimation) using the embeddings obtained by different RNN-ED variants on two publicly avail-

Table 5
Comparisons of the best results on the CMAPSS test engines (*text_FD001.txt*) in terms of the score value and accuracy obtained by each model in this work and some reported results in the literature (Note that the lower the score and the higher the accuracy, the better the model performance for RUL estimations).

	Researchers	Approaches	Score	Accuracy (%)
Deep learning-based	Ramasso [36]	RULCLIPPER	216	67
	Gugulothu et al. [8]	GRU-ED	219	59
	Malhotra et al. [7]	LSTM-Recon	256	67
	This work	Mode #1	248	71
		Mode #2	224	72
		Mode #3	245	70
		Mode #4	244	72
Non-deep learning based	Khelif et al. [37]	SV regression	449	70
	Malinowski et al. [38]	Shapelet	652	
	Khelif et al. [39]	IBL		54
	Zhang et al. [40]	MODBNE	334	
		ELM	523	
		MLP	561	

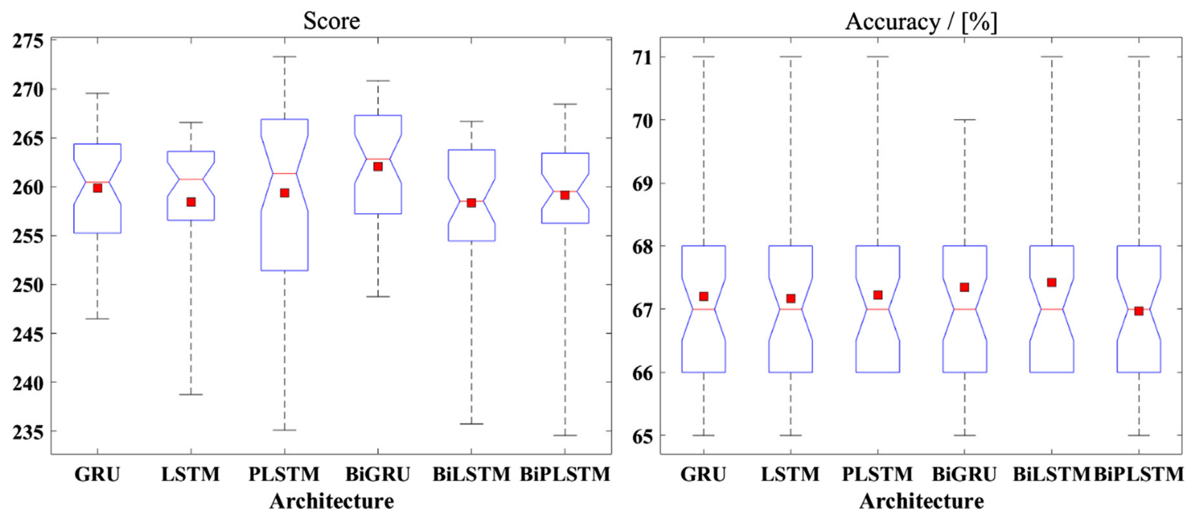


Fig. 7. The test engine RUL estimation performance of the 6 RNN-ED architectures via random search strategy in terms of the score value and accuracy for the best 20% trials.

able datasets, and compare the best results obtained by each mode of RNN-ED with those reported in the literature via a random search strategy to find the optimal values of hypermeters from a total of 200 trials each. In addition, we compare the general performance among the 4 modes of RNN-ED and 6 architectures of RNN-ED separately on the two studied cases, the results of which shed light on the significance of embeddings obtained from various RNN-ED variants on the general time series classification and machine health monitoring problems.

Our study presents two main highlights. The first is to address the necessity of reversing the output order (which was motivated by the reversing trick of seq2seq models in the sequence learning field) when training the RNN-ED as a feature extractor for time series. We demonstrate that the RNN-ED can learn robust embeddings for time series whether the output/input order is reversed or not. Thus, we challenge the commonly held assumption that the order of output sequence needs to be reversed when training RNN autoencoder as a feature extractor particularly for time series data. The second refers to performance comparisons among some common RNN-ED architectures for time series classification and machine health monitoring problems. We first observe that only the RNN architectures with gating mechanism to deal with the vanishing gradient problem can achieve the functionality of encoding for time series. Thus, we investigate three common gated architectures and their bidirectional counterparts. We found that none of the three gated architectures we studied shows significantly improved performance than the others on the two studied cases. The bidirectional RNN autoencoders yield slightly better performance than their unidirectional counterparts. In the future work, we intend to extend the evaluations and comparisons of the RNN-EDs on more data mining tasks using some other public or experimental datasets in order to fully explore their functions as unsupervised feature extractors for time series data.

CRediT authorship contribution statement

Wennian Yu: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Visualization. **Il Yong Kim:** Supervision, Project administration, Funding acquisition, Writing - review & editing. **Chris Mechefske:** Conceptualization, Supervision, Funding acquisition, Resources, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This project was financially supported by the Natural Sciences and Engineering Research Council of Canada (Grant number: RGPIN/05922-2014).

References

[1] M. Längkvist, L. Karlsson, A. Loutfi, A review of unsupervised feature learning and deep learning for time-series modeling, Pattern Recognit. Lett. 42 (2014) 11–24, https://doi.org/10.1016/j.patrec.2014.01.008.
[2] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.A. Muller, Deep learning for time series classification: a review, Data Min. Knowl. Discov. 33 (2019) 917–963, https://doi.org/10.1007/s10618-019-00619-1.
[3] A. Bagnall, J. Lines, A. Bostrom, J. Large, E. Keogh, The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances, Data Min. Knowl. Discov. 31 (2017) 606–660, https://doi.org/10.1007/s10618-016-0483-9.
[4] L. Ye, E. Keogh, Time series shapelets: a new primitive for data mining, in: Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., 2009: pp. 947–956. https://doi.org/10.1145/1557019.1557122.
[5] W. Yu, C. Mechefske, I.I.Y. Kim, Time Series Reconstruction Using a Bidirectional Recurrent Neural Network based Encoder-Decoder Scheme, in: AIAC - 18th Aust. Int. Aerosp. Congr. (AIAC2018); HUMS – 11th Def. Sci. Technol. Int. Conf. Heal. Usage Monit. (HUMS 2019); ISSFD – 27th Int. Symp. Sp. Flight Dyn., Melbourne: Engineers Australia, Royal Aeronautical Society., Melbourne, 2019, pp. 876–884. https://search.informit.com.au/documentSummary;dn=324293936594299;res=IELENG.
[6] P. Malhotra, V. TV, L. Vig, P. Agarwal, G. Shroff, TimeNet: Pre-trained deep recurrent neural network for time series classification, ArXiv Prepr. ArXiv:1706.08838. (2017). http://arxiv.org/abs/1706.08838.
[7] P. Malhotra, V. TV, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, G. Shroff, Multi-sensor prognostics using an unsupervised health index based on LSTM encoder-decoder, ArXiv Prepr. ArXiv1608.06154. (2016). http://arxiv.org/abs/1608.06154.
[8] N. Gugulothu, V. TV, P. Malhotra, L. Vig, P. Agarwal, G. Shroff, Predicting Remaining Useful Life using Time Series Embeddings based on Recurrent Neural Networks, ArXiv Prepr. ArXiv1709.01073. (2017). https://arxiv.org/abs/1709.01073.
[9] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, G. Shroff, LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection, ArXiv Prepr. ArXiv1607.00148 (2016).
[10] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, Adv. Neural Inf. Process. Syst. (2014) 3104–3112, https://doi.org/10.1007/s10107-014-0839-0.
[11] S. Hochreiter, J. Urgen Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735.
[12] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, ArXiv Prepr. ArXiv1406.1078. (2014). http://arxiv.org/abs/1406.1078.
[13] K. Greff, R.K. Srivastava, J. Koutnik, B.R. Steunebrink, J. Schmidhuber, LSTM: a search space odyssey, IEEE Trans. Neural Networks Learn. Syst. 28 (2017) 2222–2232, https://doi.org/10.1109/TNNLS.2016.2582924.
[14] N. Srivastava, E. Mansimov, R. Salakhutdinov, Unsupervised Learning of Video Representations using LSTMs, in: Int. Conf. Mach. Learn., 2015, pp. 843–852.
[15] W. Yu, I.Y. Kim, C.K. Mechefske, Remaining useful life estimation using a bidirectional recurrent neural network based autoencoder scheme, Mech. Syst. Signal Process. 129 (2019) 764–780, https://doi.org/10.1016/j.ymssp.2019.05.005.
[16] R. Jozefowicz, W. Zaremba, I. Sutskever, An empirical exploration of recurrent network architectures, in: Int. Conf. Mach. Learn., 2015, pp. 2342–2350. https://doi.org/10.1109/CVPR.2015.7298761.
[17] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science (80-.). 313 (2006) 504–508. https://doi.org/10.1126/science.1127647.
[18] J. Elman, Finding structure in time, Cogn. Sci. 14 (1990) 179–211.
[19] M.I. Jordan, Serial order: a parallel distributed processing approach, Adv. Psychol. 121 (1997) 471–495.
[20] F.A. Gers, J. Schmidhuber, Recurrent nets that time and count, in: Proc. IEEE-INNS-ENNS Int. Jt. Conf. Neural Networks. IJCNN 2000. Neural Comput. New Challenges Perspect. New Millenn., 2000, pp. 189–194. https://doi.org/10.1109/ijcnn.2000.861302.
[21] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, ArXiv Prepr. ArXiv1412.3555. (2015), https://doi.org/10.1109/ICORR.2015.7281186.
[22] K.A. Lopora, Case Western Reserve University Bearing Dataset, (2012). http://csegroups.case.edu/ bearingdatacenter/home (accessed February 21, 2020).
[23] W.A. Smith, R.B. Randall, Rolling element bearing diagnostics using the Case Western Reserve University data: a benchmark study, Mech. Syst. Signal Process. 64–65 (2015) 100–131, https://doi.org/10.1016/j.ymssp.2015.04.021.
[24] R. Zhao, D. Wang, R. Yan, K. Mao, F. Shen, J. Wang, Machine health monitoring using local feature-based gated recurrent unit networks, IEEE Trans. Ind. Electron. 65 (2017) 1539–1548, https://doi.org/10.1109/TIE.2017.2733438.
[25] M.L. Zhang, Z.H. Zhou, A review on multi-label learning algorithms, IEEE Trans. Knowl. Data Eng. 26 (2014) 1819–1837, https://doi.org/10.1109/TKDE.2013.39.
[26] A. Saxena, D. Simon, Turbofan Engine Degradation Simulation Data Set, NASA Ames Progn. Data Repos. (2008). http://ti.arc.nasa.gov/project/prognostic-data-repository (accessed April 22, 2019).
[27] A. Saxena, K. Goebel, D. Simon, N. Eklund, Damage propagation modeling for aircraft engine run-to-failure simulation, 2008 Int, Conf. Progn. Heal. Manag. PHM 2008 (2008), https://doi.org/10.1109/PHM.2008.4711414.
[28] E. Ramasso, A. Saxena, Review and Analysis of Algorithmic Approaches Developed for Prognostics on CMAPSS Dataset, in: Annu. Conf. Progn. Heal. Manag. Soc. 2014., Fort Worth, TX, USA., 2014.
[29] C.W. Hsu, C.C. Chang, C.J. Lin, A Practical Guide to Support Vector Classification, 2003. http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

[30] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (2012) 281–305, https://doi.org/10.1162/153244303322533223.
[31] M. Berglund, T. Raiko, M. Honkala, L. Karkkainen, A. Vetek, J. Karhunen, Bidirectional recurrent neural networks as generative models, Adv. Neural Inf. Process. Syst. (2015) 856–864.
[32] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778. https://doi.org/10.1109/CVPR.2016.90.
[33] T. Wang, J. Yu, D. Siegel, J. Lee, A similarity-based prognostics approach for remaining useful life estimation of engineered systems, 2008 Int. Conf. Progn. Heal. Manage. (2008) 1–6.
[34] .B. Coble, J.W. Hines, Prognostic algorithm categorization with PHM challenge application, in: 2008 Int. Conf. Progn. Heal. Manag. (2008) 1–11. https://doi.org/10.1109/PHM.2008.4711456.
[35] X. Li, Q. Ding, J.Q. Sun, Remaining useful life estimation in prognostics using deep convolution neural networks, Reliab. Eng. Syst. Saf. 172 (2018) 1–11, https://doi.org/10.1016/j.ress.2017.11.021.
[36] E. Ramasso, Investigating computational geometry for failure prognostics in presence of imprecise health indicator: results and comparisons on C-MAPPS datasets, in: 2nd Eur. Confernce Progn. Heal. Manag. Soc. (2014) 1–13. https://archivesic.ccsd.cnrs.fr/UNIV-BM/hal-01144999.
[37] R. Khelif, B. Chebel-Morello, S. Malinowski, E. Laajili, F. Fnaiech, N. Zerhouni, Direct remaining useful life estimation based on support vector regression, IEEE Trans. Ind. Electron. 64 (2017) 2276–2285.
[38] S. Malinowski, B. Chebel-morello, N. Zerhouni, Remaining useful life estimation based on discriminating shapelet extraction, Reliab. Eng. Syst. Saf. 142 (2015) 279–288, https://doi.org/10.1016/j.ress.2015.05.012.
[39] R. Khelif, S. Malinowski, B. Chebel-Morello, N. Zerhouni, RUL prediction based on a new similarity-instance based approach, in, IEEE Int. Symp. Ind. Electron. (2014) 2463–2468.
[40] C. Zhang, P. Lim, A.K. Qin, K.C. Tan, Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics, IEEE Trans. Neural Networks Learn. Syst. 28 (2017) 2306–2318, https://doi.org/10.1109/TNNLS.2016.2582798.