Chinese Journal of Mechanical Engineering

ORIGINAL ARTICLE Open Access

Practical Options for Adopting Recurrent Neural Network and Its Variants on Remaining Useful Life Prediction

Youdao Wang, Yifan Zhao* and Sri Addepalli

Abstract

The remaining useful life (RUL) of a system is generally predicted by utilising the data collected from the sensors that continuously monitor different indicators. Recently, different deep learning (DL) techniques have been used for RUL prediction and achieved great success. Because the data is often time-sequential, recurrent neural network (RNN) has attracted significant interests due to its efficiency in dealing with such data. This paper systematically reviews RNN and its variants for RUL prediction, with a specific focus on understanding how different components (e.g., types of optimisers and activation functions) or parameters (e.g., sequence length, neuron quantities) affect their performance. After that, a case study using the well-studied NASA's C-MAPSS dataset is presented to quantitatively evaluate the influence of various state-of-the-art RNN structures on the RUL prediction performance. The result suggests that the variant methods usually perform better than the original RNN, and among which, Bi-directional Long Short-Term Memory generally has the best performance in terms of stability, precision and accuracy. Certain model structures may fail to produce valid RUL prediction result due to the gradient vanishing or gradient exploring problem if the parameters are not chosen appropriately. It is concluded that parameter tuning is a crucial step to achieve optimal prediction performance .

Keywords: Remaining useful life prediction, Deep learning, Recurrent neural network, Long short-term memory, Bi-directional long short-term memory, Gated recurrent unit

1 Introduction

Remaining useful life (RUL) prediction is an engineering discipline that works on the prediction of the future state or response of a given system based on synthesis observations, calibrated mathematical models, and simulations [1]. It generally refers to the study of predicting the specific time at which the system or the component will no longer be able to have its intended functional performance. Salunkhe et al. [2] regard RUL as the time left before observing a failure. Okoh et al. [3] define RUL as the time remaining for a component to perform its functional capabilities before failure. It is of great importance to predict the RUL of a component or a system in the industrial world, as it helps to prevent failures or accidents from happening. For example, the failure of the aircraft engine would often lead to major accidents and casualties [4]. Thus, it is essential to predict the RUL of the engine, implement maintenance accordingly and eventually prevent catastrophic failure. The degradation process of an operating device is a process of gradual deterioration and can be detected to a certain extent through the measurement of covariate variables [5]. In recent years, RUL prediction has attracted vast attention from both academic researchers and industrial operators.

RUL prediction approaches are generally catalogued into model-based (physics-based) methods, data-driven methods and hybrid models, which is a combination of the first two methods [5]. As the complex and noisy

*Correspondence: yifan.zhao@cranfield.ac.uk
School of Aerospace, Transport and Manufacturing, Cranfield University, Cranfield MK43 0AL, UK

working condition impedes the construction of the physical systems, it results in difficulties in developing the modelling of complex dynamic systems [6]. In addition, the difficulty to be updated with the online measured data, limits the effectiveness and flexibility of the physics-based models. In contrast, data-driven approaches are gaining popularity due to its quick implementation and widespread deployment of low-cost sensors and their connection to the internet, where RUL is computed through statistical and probabilistic methods by utilising historic information and routinely monitored data of the system [7]. The precondition for setting up the data-driven models for RUL prediction is the availability of the multivariate historical data about the system behaviour, which must encompass all phases of the system operation and degradation scenarios under certain operating conditions. In recent years, Artificial intelligence (AI) techniques, particularly deep learning (DL) techniques are becoming more and more attractive because of the rapid growth in the industrial Internet of Things (IoT), Big Data and increasing computing power [8]. Researchers have exploited applications of AI techniques for RUL prediction as well.

Deep learning is one of the sub-branches of machine learning, which originated from the Artificial Neural Network (ANN) and featuring multiple nonlinear processing layers. It intends to model hierarchical representations and predicts patterns behind data through building stacked multiple layers of information processing modules in hierarchical architectures. With the rapid development of computational infrastructure and the availability of a large volume of data, DL has become one of the main research topics in the field of prognostics, given its capability to capture the hierarchical relationship embedded in deep structures [9]. The published literature on DL approaches for RUL prediction mainly focus on four representative deep architectures, including Auto-encoder (AE), Deep Belief Network (DBN), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) [10]. AE and DBN are often used for the pre-training of networks. For instance, Jia et al. [11] developed a stacked denoising autoencoder (SDA) which is fed with the frequency spectra of time-series data to do the rotating machinery diagnosis. Chen et al. [12] proposed an SDA to identify the health state of certain systems with signals containing ambient noise and working condition fluctuations. Shao et al. [13] developed a deep AE based method to diagnose rotating machinery fault. Liao et al. [14] proposed to combine an enhanced Restricted Boltzmann Machine (RBM) with a novel regularisation term to automatically extract the features which are suitable for RUL prediction. Gan et al. [15] presented a hierarchical diagnosis network that combines a wavelet packet transform (WPT) and DBN for consecutive identification of bearing fault location and severity. Thus, research suggests that CNN and RNN are generally used as predictive models and have proved to outperform traditional prognosis algorithms in RUL prediction. CNN based approaches are used more in fault diagnosis and surface integration inspection [16]. RNN, on the other hand, gained much more attention and achievements in the research of RUL prediction because of its ability to accommodate time sequence data [17]. Therefore, this paper systematically reviews the applications of RNN and its variants for RUL prediction in recent years. Many novel RNN based methods have been proposed, and the performance of the RUL performance has been greatly improved. However, most of these works just focused on how to achieve a better prediction performance using a certain approach. Very few researchers paid attention to some other factors that also affect the prediction result such as the optimizer, activation function, neuron number and sequence length. Taking the optimizers as an example, they are used to shape the model into its most accurate form through futzing with the weights. To the best of our knowledge, there is no research discussing how different optimizers affect the performance of RUL prediction using DL based approaches, and what is the underlying principle to optimise the selection. To fill these research gaps, this paper not only presents an evaluation of the basic RNN and its variants on RUL prediction based on a case study in a publicly available dataset, but a specific investigation has also been carried out on how different components (e.g., types of optimisers and activation functions) or parameters (e.g., sequence length, neuron quantities) of these approaches affect the overall performance (e.g., stability, precision, accuracy) of the RUL prediction.

The remainder of this paper is organized as follows. Section 2 briefly introduces the basic conception of RNN and its variants. Section 3 presents the different optimizers that is normally used in DL. Section 4 explains how activation functions affect the training of the network and demonstrate the advantages and drawbacks of different activation functions. Section 5 presents a case study that aims to evaluate the factors that influence the performance of RUL prediction based on a publicly available dataset.

2 RNN and Its Variants

2.1 RNN

In a traditional neural network, inputs are independent, while in RNN, the front neurons pass the information to the following neurons. As illustrated in Figure 1, in contrast to a traditional feed-forward neural network, an RNN can be regarded as numerous copies of the same neural network cell, in which each cell passes the message

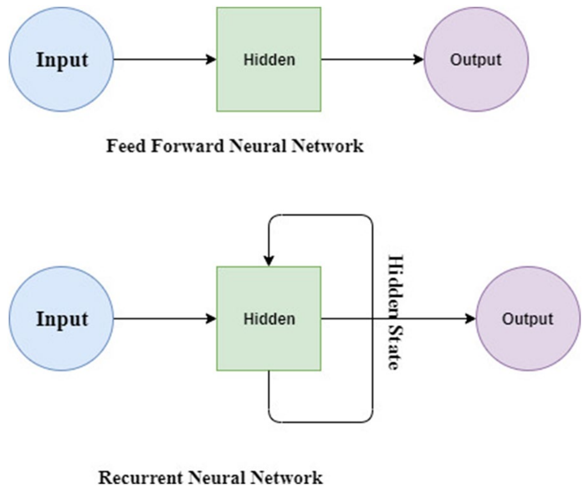

Figure 1 Basic Feed-Forward Neural Network structure and Recurrent Neural Network structure

to the next through the hidden state. In other words, the output from a recurrent neuron is connected to the next one to characterise the current system state as a function of current sensing data and the preceding system state.

In an unrolled RNN, the sensing data ($\ldots x_{t-1}$, x_t, $x_{t+1}\ldots$) are fed simultaneously into the corresponding neurons, which generate the corresponding neuron time series ($\ldots$ h_{t-1}, h_t, h_{t+1} $\ldots$). The output of a single recurrent neuron can be expressed as:

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b), \tag{1}$$

$$y_t = softmax\left(W_y h_t + c\right), \tag{2}$$

where W_x, W_h and W_y represent the weight vectors respectively. The symbol b and c denote the bias term and σ is the activation function, with the hyperbolic tangent or Relu being commonly used in RNN. y_t is the output of the recurrent neuron based on the output of the hidden state h_t, which can be referred to as a memory space containing the information of the current input and the former hidden state h_{t-1}. It is worth mentioning that all the weight vectors are shared at every step, which means that the same task is repeated at every step with different inputs and the memory is renewed accordingly.

2.2 LSTM

The main issue of the standard RNN is the gradient exploring and the gradient vanishing. These issues might happen when the network is too deep. In other words, when the number of the time step is too large, the information carried in the front neuron will be lost because no structure in a standard recurrent layer individually controls the flow of the memory itself. To solve this problem, the Long Short-Term Memory (LSTM) network, a modified structure of the recurrent cell that incorporates the standard recurrent layer along with additional "memory" control gates, has been proposed. The basic structures of RNN, LSTM and GRU are illustrated in Figure 2.

The original LSTM was developed by Hochreiter and Schmidhuber [18] when researchers discovered a vanishing and exploding gradient issue in traditional RNNs. LSTM uses storage elements to transfer information from the past output instead of having the output of the RNN cell to be a non-linear function of the weighted sum of the current input and the previous output.

In another words, instead of using a hidden state h only, LSTM adopts a cell state C to keep the long-term information as shown in Figure 3. The main concept of LSTM is utilising three gates to control the cell state C (forget gate, input gate and output gate). The forget gate is used to control the information from the previous cell state C_{t-1} to the current cell state C_t; the input gate decides how many inputs should be kept in the current cell state C_t ; and the output gate determines the output h_t from the current cell state C_t.

The output of LSTM at step t is calculated using the following equations:

$$i_t = \sigma\left(U^i x_t + W^i h_{t-1} + b_i\right), \tag{3}$$

$$f_t = \sigma\left(U^f x_t + W^f h_{t-1} + b_f\right), \tag{4}$$

$$o_t = \sigma\left(U^o x_t + W^o h_{t-1} + b_o\right), \tag{5}$$

$$\tilde{c}_t = \tanh\left(U^{\tilde{c}} x_t + W^{\tilde{c}} h_{t-1} + b_{\tilde{c}}\right), \tag{6}$$

$$C_t = C_{t-1} \cdot f_t + \underset{t}{\tilde{c}} \cdot i_t, \tag{7}$$

$$h_t = \tanh(c_t) \cdot o_t, \tag{8}$$

where W and b are the trainable weights and biases, respectively, and i, f and o represent the input gate, forget gate and output gate respectively. These three gates have the same shape with different parameters U and W, which need to be learned from the training process. The candidate state $\tilde{c}_t$ cannot be used directly. It must pass through the input gate and then be used to calculate the internal storage C_t. While C_t is not only affected by the hidden state but also by C_{t-1} which is controlled by the forget gate. Based on C_t, a layer of tanh function is applied to the output information h_t,

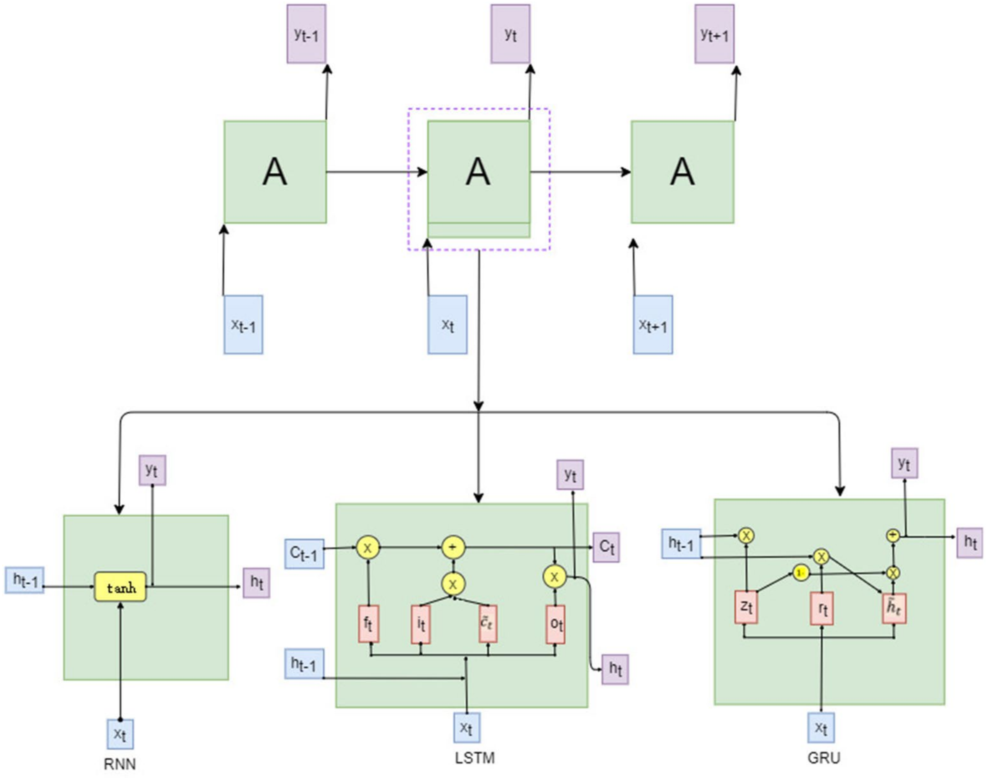

Figure 2 Basic structures of RNN, GRU and LSTM

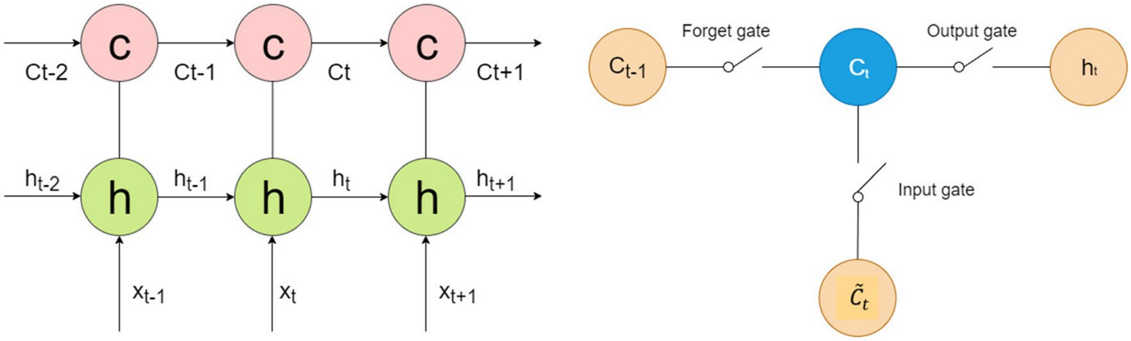

Figure 3 Cell state and three gates in LSTM

which is constrained by the output gate. The existence of the gates enables LSTM to fulfil the long-term dependencies in the sequence, and by learning the gate parameters, the network can find the appropriate internal storage. Therefore, LSTMs are naturally suited for RUL prediction tasks using sensor data with the inherent sequential nature due to their capability of remembering information over long periods. Yuan et al. [19] proposed an LSTM approach for different types of faults, where C-MPASS dataset was used as the case study. Compared to the traditional RNN, Gated Recurrent Unit LSTM (GRU-LSTM) and AdaBoost-LSTM showed improved performance in all cases. They developed a vanilla LSTM approach two years later which further improved the prediction performance significantly [20]. A multi-layer LSTM approach provided by Zheng et al. [17] investigated the hidden patterns from sensors and operational data with multiple operating conditions, fault and degradation models by combining multiple layers of LSTM cells with standard feed-forward layers. The superiority of this approach in RUL prediction was validated by three widely used data sets, C-MAPSS Data Set, PHM08 Challenge Data Set and the Milling Data Set.

2.3 GRU

The shortcoming of LSTMs is that it is usually time-consuming due to the forget gate, input gate and output gate added to the structure of the memory blocks. To address this problem, an improved structure, named Gated Recurrent Unit (GRU), was proposed [21].

GRU is the latest generation of RNN, and it looks very similar to LSTM. Instead of using the cell state, GRU uses the hidden state to transfer information. Moreover, it only has two gates (a reset gate and update gate) instead of three. Similar to the forget and input gate of LSTM, the function of the update gate is to decide what information to keep and what to throw away. The function of the reset gate is to decide what to keep from the past information.

The output of GRU at step t is calculated using the following equations:

$$z_t = \sigma\left(U^Z x_t + W^Z h_{t-1} + b_Z\right), \tag{9}$$

$$r_t = \sigma\left(U^r x_t + W^r h_{t-1} + b_r\right), \tag{10}$$

$$\tilde{h}_t = \tanh\left(U^{\tilde{h}} x_t + W^{\tilde{h}} h_{t-1} \cdot r_t + b_{\tilde{h}}\right), \tag{11}$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t, \tag{12}$$

Since there are fewer tensor operations in GRU, it runs relatively faster when training the structure than LSTM. However, the accuracy is behind LSTM due to fewer gates. Thus, when the computational resource is limited, or fast training is required, GRU could be a good option. For instance, Chen et al. [22] adopted a GRU network to predict the RUL for a complex system featured with multiple components, multiple states and a large number of parameters.

2.4 Bi_directional LSTM

In recent years, there is another variant of RNN called Bi_directional LSTM (Bi_LSTM) that can be seen frequently in literature. The Bi-directional LSTM is proposed with the information flowing back to the former LSTM cells. The forward flow of information can discover the system variation, and it flows back to smooth the predictions as illustrated in Figure 4. The outputs of the forward path and the backward path are then concatenated. The governing equations of Bi-directional LSTM can be presented as:

$$h_i^1 = f\left(U^1 \cdot x_i + W^1 \cdot h_{i-1}\right), \tag{13}$$

$$h_i^2 = f\left(U^2 \cdot x_i + W^2 \cdot h_{i-1}\right), \tag{14}$$

$$y_i = softmax\left(V \cdot \left[h_i^1; h_i^2\right]\right), \tag{15}$$

where Eq. (13) refers to the forward path and Eq. (14) refers to the backward path, y_i is the output of the Bi-directional LSTM obtained by fusing the results from both directional paths.

As for the application, Zhao et al. [23] presented an integrated approach of CNN and bi-directional LSTM for machining tool wear prediction named Convolutional Bi-directional Long Short-Term Memory (CBLSTM) networks. CNN was firstly used to extract local robust features from the sequential input. Then, Bi-directional LSTM was utilised to encode temporal information. The proposed CBLSTM's capability of predicting the RUL of actual tool wear based on raw sensory data was verified with a real-life tool wear test. Zhang et al. [24] presented a Bi-directional LSTM network to discover the

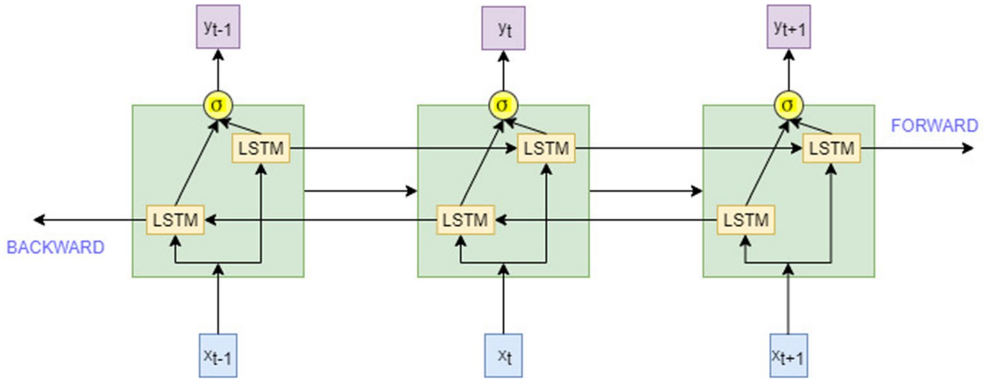

Figure 4 A Bi-directional LSTM Structure

underlying patterns embedded in time-series to track the system degradation. The Bi-directional LSTM network was implemented to track the variation of the health index, and the RUL was predicted by the recursive one-step ahead method. Elsheikh et al. [25] built a Bidirectional Handshaking LSTM (BHLSTM) network for RUL prediction, where short sequences of monitored observations were given with random initial wear. This method was able to predict the RUL with a random start, which makes it more suitable for real-world application as the initial condition of physical systems is usually unknown, especially in terms of its manufacturing deficiencies.

3 Optimizer

Gradient descent by far is the most commonly used way to optimise neural network [26]. It is an iterative optimization algorithm used to find the values of parameters or coefficients of a function that minimizes a cost function. Although various algorithms have been developed to optimize gradient descent, they are usually used as black-box optimizers because it is hard to figure out the practical explanations of their strengths and weaknesses.

Different in how much data used to compute the gradient of the objective function, the gradient descent variants are classified into two categories: batch gradient descent (BGD) and stochastic gradient descent (SGD). BGD is guaranteed to converge to a global minimum for convex error surfaces and a local minimum for non-convex surfaces. However, BGD can be very time-consuming because it needs to calculate the gradients for the whole dataset to perform just one update and thus it is intractable for datasets that do not fit in memory. In addition, BGD cannot be used to update the model online. In contrast, SGD performs one update at a time, and thus it will not have any redundant computations for large datasets as BGD does. As a result, SGD is usually much fast than BGD. Meanwhile, it can be used to learn the model online. The drawback of SGD is that the frequent updates with a high variance would lead to a heavy fluctuation to the objective function. While if the learning rate is slowly decreased over time, SGD shows the same convergence behaviour as BGD, it almost certainly converges to a local or the global minimum for non-convex optimization.

Although SGD can often lead to good convergence, few challenges need to be addressed. For instance, it is difficult to determine a proper learning rate and an annealing schedule, or it is hard to update features to a different extent avoiding suboptimal minima. Ruder [26] outlines some algorithms that are widely used by the deep learning community which can deal with these challenges includes Momentum, Nesterov accelerated gradient, Adagrad, Adadelta, RMSprop, Adam, AdamMax and Nadam. Ruder also stated that Adagrad, Adadelta, RMSprop and Adam can all significantly improve the robustness of SGD and do not need much manual tuning of the learning rate. These four optimizers are therefore selected and discussed in more detail in this paper.

3.1 Adagrad

Adagrad is a gradient-based optimizer that adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent updates. Thus, it is very suitable for sparse data. It uses a different learning rate for every parameter θ_i at every time step t, so the gradient of the objective function $g_{t,i}$ regarding the parameter θ_i at time step t is written as:

$$g_{t,i} = \nabla_{\theta_t} J\left(\theta_{t,i}\right), \tag{16}$$

The SGD updates for every parameter θ_i at each time step t following equation:

$$\theta_{t+1,\,i} = \theta_{t,i} - \eta \cdot g_{t,i}. \tag{17}$$

Adagrad modifies the general learning rate η at each time step t for every parameter θ_i based on the past gradients:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \in}} \cdot g_{t,i}, \tag{18}$$

where $G_t \in R^{d \times d}$ is a diagonal matrix where each diagonal element i is the sum of the squares of the gradients regarding the parameter θ_i at time step t, $\in$ is a smoothing term used to avoid division by zero.

One of the main advantages of Adagrad is that it is not required to manually tune the learning rate. The default value is set as 0.01. The main drawback of this optimizer is that its accumulation of the squared gradients in the denominator would result in the learning rate to shrink and become infinitesimally small, which means that at a certain point, the algorithm can no longer acquire additional knowledge.

3.2 Adadelta

To reduce the monotonically decreasing learning rate, an extension optimizer of Adagrad has been promoted, named Adadelta. It uses a fixed-size window of accumulated past gradients instead of accumulating all past squared gradients. The sum of the gradient is recursively defined as a decaying average of all past squared gradients. Thus, the running average of the squared gradients of the objective function at time step t depends on the previous average and the current gradient:

$$\mathrm{E}[g^2]_t = \gamma \mathrm{E}[g^2]_{t-1} + (1-\gamma) g_t^2, \tag{19}$$

where γ is the fraction of the update vector of the past time step to the current update vector, which is normally set to 0.9 [26].

The SGD update for parameter $\Delta\theta_t$ at each time step t therefore becomes:

$$\Delta\theta_t = -\eta \cdot g_{t,i}, \tag{20}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t. \tag{21}$$

And according to the update rule, through simply replacing the diagonal matrix G_t with the decaying average of past squared gradients $E[g^2]_t$, the parameter update vector of Adadelta can be derived as:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{\mathrm{E}\left[g^2\right]_t + \in}} \cdot g_{t,i}. \tag{22}$$

As $\sqrt{E\left[g^2\right]_t + \in}$ is the root mean squared (RMS) error criterion of the gradient, it can then be written as:

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} \cdot g_{t,i}. \tag{23}$$

Since the update should have the same hypothetical units as the parameter, the exponentially decaying average of the squared parameter should be used:

$$E\left[\Delta\theta^2\right]_t = \gamma E\left[\Delta\theta^2\right]_{t-1} + (1-\gamma)\Delta\theta_t^2, \tag{24}$$

$$\Delta\theta_t = -\frac{\mathrm{RMS}[\Delta\theta]_{t-1}}{RMS[g]_t} \cdot g_t, \tag{25}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t. \tag{26}$$

Based on the update rule of Adadelta, there is no need for setting a default learning rate.

3.3 RMSprop

RMSprop is an adaptive learning rate method designed for neural networks which have been growing in popularity in recent years. Similar to Adadelta, the central idea of RMSprop is to keep the moving average of the squared gradients for each weight and then divide the gradient by square root of the mean square. However, a good default value of decay parameter γ and learning rate are set to 0.9 and 0.001:

$$\mathrm{E}\left[g^2\right]_t = 0.9\mathrm{E}\left[g^2\right]_{t-1} + 0.1g_t^2, \tag{27}$$

$$\theta_{t+1} = \theta_t - \frac{0.001}{\sqrt{\mathrm{E}\left[g^2\right]_t + \in}} \cdot g_{t,i}. \tag{28}$$

3.4 Adam

Another method that computes adaptive learning rates for each parameter was named Adaptive Moment Estimation (Adam) [27]. Adam not only stores an exponentially decaying average of past squared gradients v_t, but also keeps an exponentially decaying average of past gradients m_t, as shown in Eqs. (29) and (30):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t, \tag{29}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2, \tag{30}$$

where m_t refers to the estimate of the first moment (the mean) of the gradients and v_t refers to the second moment (the uncentered variance) of the gradients. As the initial value of m_t and v_t are vectors of zeros, it is observed that when the decay rates are small during the initial time, they are biased towards zero. The biases are counteracted by computing bias-corrected first and second moment estimates:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \tag{31}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}. \tag{32}$$

Therefore, the update rule of Adam can be derived as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \in}} \cdot \hat{m}_t. \tag{33}$$

The proposed values for β_1, β_2 and $\in$ are 0.9, 0.999 and 10^{-8}, respectively.

4 Activation Function

The activation function is a function working on a neuron in an ANN and mapping the input of the neuron to the output. More specifically, each neuron node in the neural network adapts the output of the neuron in the upper layer as the input and passes it to the next layer (hidden layer or output layer). Thus, the activation function refers to the functional relationship between the output of the upper node and the input of the lower node in the multilayer neural network. Without an activation function, the input of each layer will be linear to the output of the upper layer. No matter how many layers the neural network has, the output is just a linear combination of the input, which is similar to the original perceptron. To enable the neural network arbitrarily to any nonlinear function, the activation function introduces a nonlinear factor to the neuron. The nonlinear activation functions allow the network to learn complex form data and complex function mappings that represent nonlinearity between input and output.

There are three types of activation functions normally used in the deep learning area: tanh & sigmoid, ReLU and swish. In this section, the basic mathematical expression of these three types of activation functions is reviewed with their advantages and drawbacks. The expression of these activation functions and their variants are demonstrated in Figure 5.

4.1 Sigmoid & Tanh

Sigmoid function, expressed in Eq. (34), also known as Logistic function, is normally used for the output of the hidden layer neurons:

$$\sigma(x) = \frac{1}{1+e^{-x}}, \; x \in (0,1), \tag{34}$$

The advantage of Sigmoid function is that the output of the activation function is limited between 0 and 1, which results in a stable optimization and thus good to be used as the output layer. The drawback is that the function could be very insensitive to small changes in input when a variable takes a very large positive or negative value. During the backpropagation, the weight will hardly be updated when the gradient gets close to zero. Therefore, the gradient will disappear, and the network will be able to complete its training. In addition, the output of sigmoid function is not zero mean, which leads to the input of neurons in the back layer being non-zero mean, and then affects the gradient. Besides, due to the exponential form in the sigmoid function, the computational complexity is very high.

Tanh function, expressed in Eq. (35), is also called the hyperbolic tangent function:

$$f(x) = \tanh x = \frac{\sinh x}{\cosh x} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}},$$

$$x \in (-1,1) \tag{35}$$

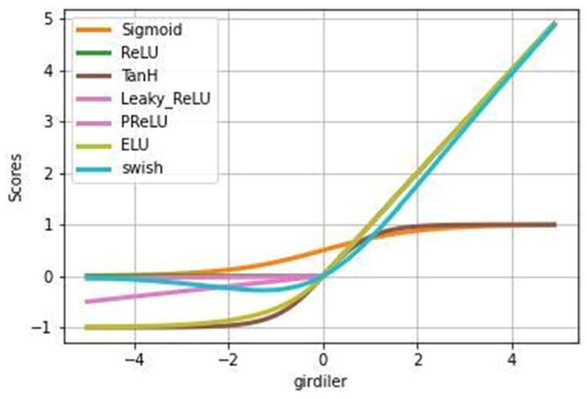

Figure 5 Different activation functions

Tanh function is the translation and contraction of sigmoid function: tanh(x)=2·σ(2x)−1. Tanh function often outperforms sigmoid in practice because its output is zero mean. Nevertheless, it still suffers from gradient saturation and computational complexity.

4.2 ReLU

Rectification of linear unit (ReLU) is the most commonly used deep learning neural network activation function. It is the default activation function for most of the feed-forward neural networks. The ReLU function is written as:

$$f(x) = \max(0, x) \tag{36}$$

The advantage of ReLU function is that the SGD algorithm converges faster than sigmoid or tanh. When the weight is larger than zero, there are no problems like gradient saturation and gradient disappearance. Since there is no need to carry out the exponential operation, the computational complexity is relatively low. A threshold is needed for achieving the activation value. The limitation of ReLU function is that the output is not zero mean either. Besides, the Dead ReLU Problem will occur when the weight is in the negative field. During the training, when x is less than zero, the gradient of the current neuron and the neurons after it is always zero. In other words, it will no longer respond to any data and the corresponding parameters would never be updated. To solve this problem, Leaky ReLU, Parametric Rectified Linear Unit (PReLU) and Exponential Linear Unit (ELU) were introduced.

Leaky ReLU function:

$$f(x) = \max(0.01x, x). \tag{37}$$

PReLU function:

$$f(x) = \max(\alpha x, x). \tag{38}$$

ELU:

$$f(x) = \begin{cases} \alpha\left(e^{x}-1\right), & x \le 0, \\ x, & x > 0. \end{cases} \tag{39}$$

The Leaky ReLU uses a small value of 0.01 to initialize the neuron so that the ReLU function can be activated in the negative region. The difference between Leaky ReLU and PReLU is that α of PReLU function is learned through backpropagation. ELU has all the advantage of ReLU and no Dead ReLU Problem. It can make the average activation mean value of neurons close to zero and at the same time, which suggests that it is more robust to noise. However, because of the exponential form, the calculation complexity is relatively higher.

4.3 Swish

Swish is a self-gated activation function proposed by Prajit et.al. [28], who attempted to use an automated search technique to find novel activation functions to replace the ReLU function without changing the network architecture. By a combination of exhaustive and reinforcement learning-based search, they found a number of novel promising activation functions and named the best one of them as Swish.

The Swish function can be written as:

$$f(x) = x \cdot \mathrm{sigmoid}(\beta x), \tag{40}$$

where β is a constant or trainable parameter.

5 Case Study

5.1 Benchmark Dataset Overview

The case study focuses on the investigation of the influence of various practical options of optimizers, activation functions and other parameters like sequence length and neuron number when adopting RNN and its variants on RUL prediction. We selected the NASA's Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dataset, aiming at modelling the damage propagation of aircraft gas turbine engines [29]. This engine simulator produced four datasets which are consisted of three operational condition indicators. Each subset has different numbers of engines with varied operational cycles.

In the dataset, engine profiles were simulated with different initial degradation conditions. The maintenance was not considered during the simulation. The dataset includes one training set and one testing set for each engine. The training set consists of the historical run-to-failure measurement records of the engines from 21 on-board sensors. The objective is to predict the RUL of each engine based on the given sensor measurements. The information of the four subsets is listed in Table 1. Specifically, FD001 refers to the engine failure arising from the high-pressure compressor under a single operating condition. FD002 refers to the engine failure from the high-pressure compressor under six operation conditions. FD003 refers to the engine failure from both high-pressure compressor and fan under a single operating condition. FD004 refers to the engine failure from both high-pressure compressor and fan under six operation conditions. In this study case, FD001 was used because the data volume is relatively small, therefore it is more time-efficient to do the test.

5.2 Data pre-processing

The raw sensor data were normalised to [0, 1]. No dimension reduction and feature extraction have been taken place in this case study and the entire sensor data stack was used as inputs for training. In addition, since there is no target output in raw datasets, the RUL was labelled at every cycle for each sample before training the models.

5.3 Performance Evaluation

In this case study, the mean square error (MSE) was used to evaluate the performance of the trained neural networks. The mathematical expression is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} d_i^2, \tag{41}$$

where n is the total number of true RUL targets in the related test set and d_i refers to the difference between the true RUL and the predicted RUL.

The RNN algorithm and its variants were tested with the dataset FD001, and three different layer structures for each method were used. Each algorithm and structure have been tested five times to achieve the statistical result, which was illustrated in the form of a box chart. The results were presented and discussed according to four main factors: optimizers, activation functions, neuron numbers and sequence lengths against three assessment criteria: stability, precision and accuracy. The ranks for precision and accuracy for these four factors will be presented. As for the stability, if the network can produce a reliable result, it will be marked as 1, otherwise, it will be marked as 0.

5.3.1 Optimizers

Different neural network structures were tested with the fixed activation function of ReLU, the neuron number of 128, the sequence length of 50, four different optimizers including Rmsprop, Adam, AdamGrad and AdamDelta.

Table 1 C-MAPSS dataset

Dataset	FD001	FD002	FD003	FD004
Data for training	100	260	100	249
Data for test	100	259	100	248
Operating conditions	Single	Multiple	Single	Multiple
Fault conditions	Compressor	Compressor	Compressor & fan	Compressor & fan

The prediction results are displayed using box plots so that the stability, precision and accuracy of these optimizers can be evaluated.

As indicated in Figure 6, gradient exploring or gradient vanishing took place when adopting AdaGrad in RNN_2LAYERS, RNN_3LAYERS, LSTM_2LAYERS, LSTM_3LAYERS and GRU_3LAYERS and AdaDelta in RNN_LAYERS, Bi_LSTM_3LAYERS. This observation suggests that RMSprop and Adam are less sensible to the parameters than AdaGrad and AdaDelta, which means they are more workable in this case. More specifically, AdaGrad and AdaDelta are more likely to lose their stability when the network gets more complicated. In terms of accuracy, generally speaking, AdamGrad can help to achieve the most accurate prediction result in most network structures, regardless of the stability. As for Rmsprop, the change in the structure layers would make a great difference to the prediction performance. In contrast, this influence can hardly be seen when adopting Adam and AdaDelta as the optimizers. As for the precision, Rmsprop has the worst performance among these four optimizers where the other three can all produce relatively precise outcomes.

The assessment of the four optimizers have been made for all network structures such as the example set in Table 2, and all the optimal optimizers have been summarized in Table 3. In this case, AdaGrad can be regarded as the optimal optimizer for most of the network structures.

5.3.2 Activation Functions

In this section, the evaluation of five activation functions is performed with the fixed optimizer (Adam), neuron number (128) and sequence length (50). Both Sigmoid and Tanh functions have also been tested, but these two activation functions were found to be greatly

Table 2 Assessment of different optimizers using network structure RNN-1LAYER

Optimizers	Stability	Precision	Accuracy	Assessment
RMSprop	1	4	4	
Adam	1	3	3	
AdaGrad	1	1	1	√
AdaDelta	1	2	2	

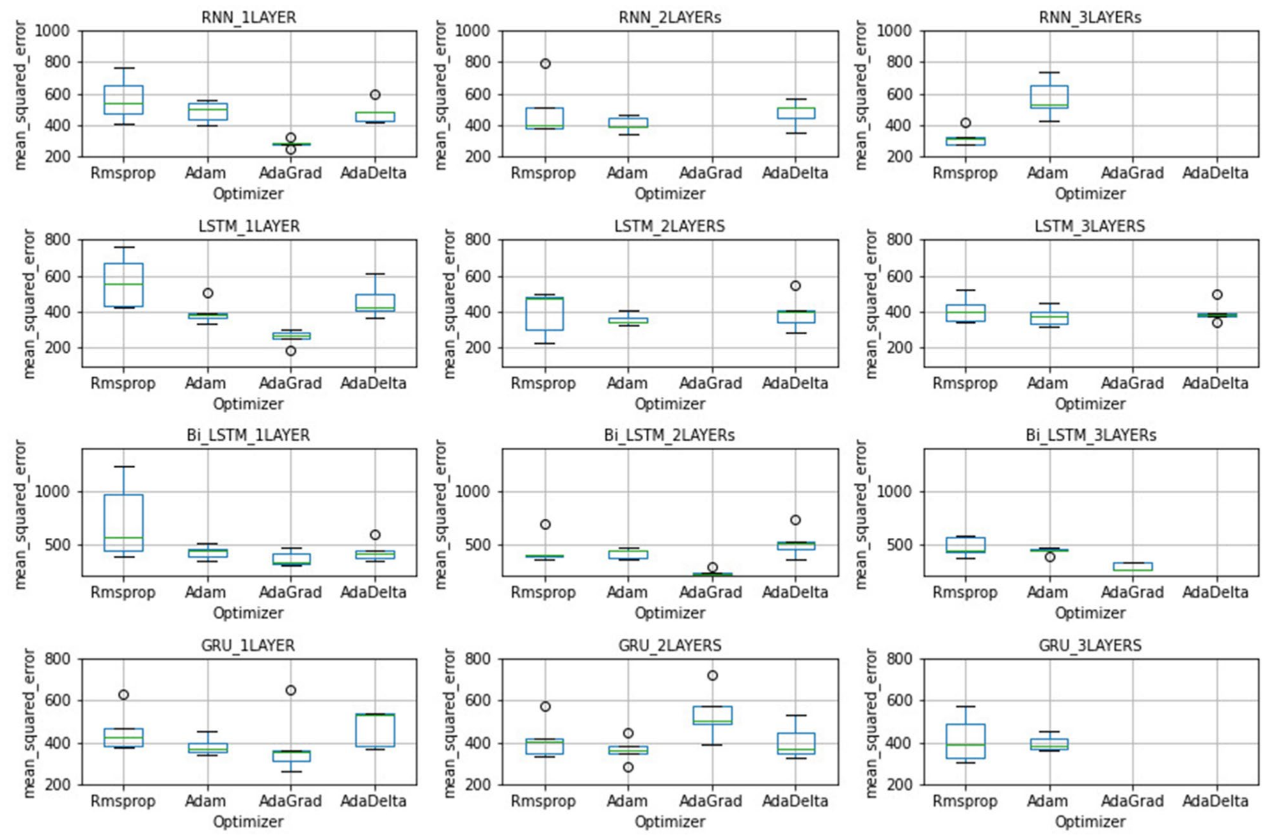

Figure 6 Performance of different optimizers on the testing dataset

Table 3 Optimal optimizers for different algorithms

Algorithms	Optimal optimizers
RNN_1LAYER	AdaGrad
RNN_2LAYERS	Adam
RNN_3LAYERS	Adam
LSTM_1LAYER	AdaGrad
LSTM_2LAYERS	Adam
LSTM_3LAYERS	Adam, AdaDelta
Bi_LSTM_1LAYER	Adam, AdaGrad, AdaDelta
Bi_LSTM_2LAYERS	AdaGrad
Bi_LSTM_3LAYERS	AdaGrad
GRU_1LAYER	Adam, AdaGrad
GRU_2LAYERS	AdaGrad
GRU_3LAYERS	AdaGrad

affected by the gradient vanishing and gradient exploring problem. Therefore, these two functions are not discussed in this section. As demonstrated in Figure 7, the performance of ReLU, Leaky_ReLU, PReLU and ELU is quite similar, and they are generally better than Swish in both precision and accuracy. However, gradient exploring, or gradient vanishing occurred when adopting ReLU, PReLU and ELU in RNN_1LAYER and GRU_3layer, which suggests that Swish and Leaky_ReLU are more stable than these three activation functions. The Optimal activation functions in this case for different algorithms are listed in Table 4.

5.3.3 *Sequence Length*

In this section, the impact of different sequence length on the prediction result has been compared with a fixed optimizer (Adam), activation function (ReLU) and neuron number (128). As indicated in Figure 8, generally the longer the sequence length uses, the better performance the algorithms achieved. In this case, gradient vanishing happened when adopting GRU_3LAYERS network structure which suggests that the choice of the sequence length may also affect the workability of GRU. The optimal sequence length for different algorithms is listed in Table 5 considering the workability, precision and accuracy.

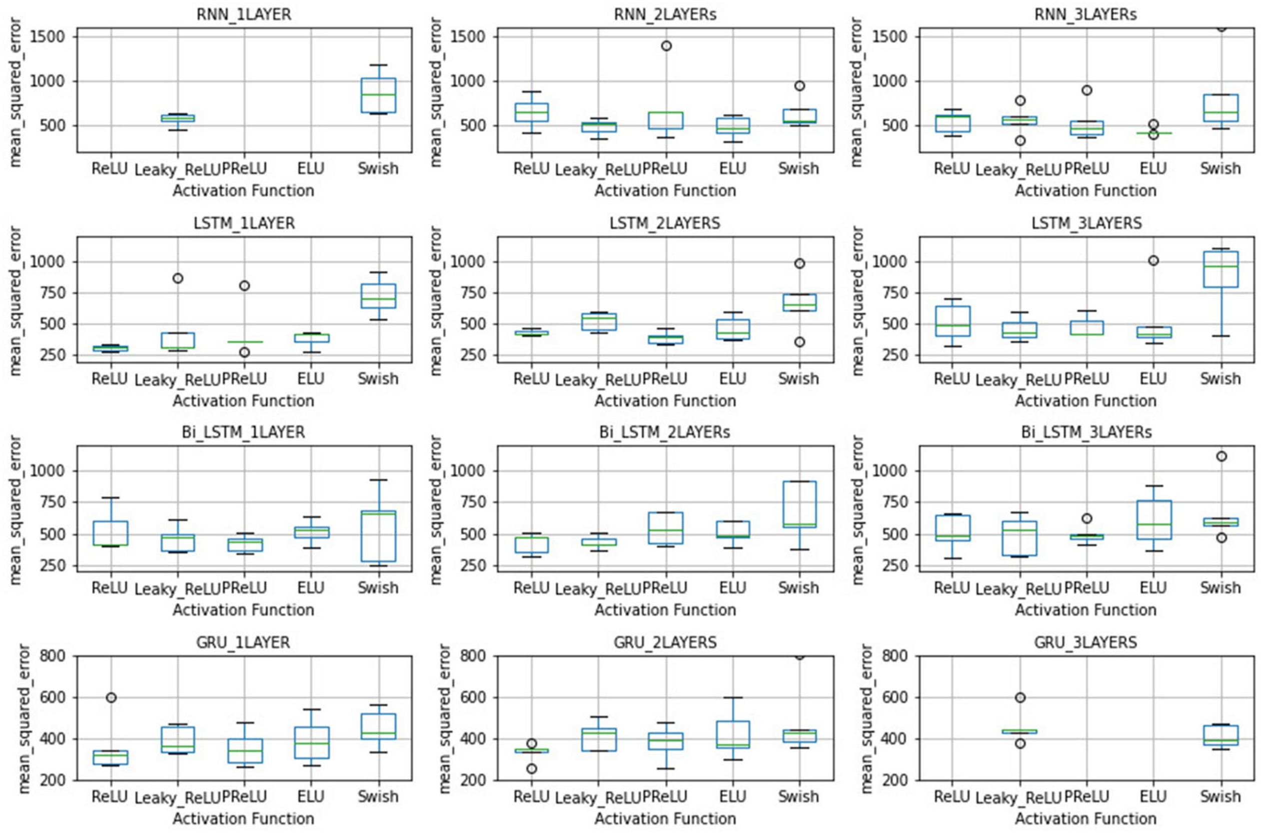

Figure 7 Performance of different activation functions on the test dataset

Table 4 Optimal activation functions for different algorithms

Algorithms	Optimal activation functions
RNN_1LAYER	Leaky_ReLU
RNN_2LAYERS	Leaky_ReLU, ELU
RNN_3LAYERS	ELU
LSTM_1LAYER	ReLU
LSTM_2LAYERS	PReLU
LSTM_3LAYERS	ELU
Bi_LSTM_1LAYER	PReLU
Bi_LSTM_2LAYERS	ReLU, Leaky_ReLU
Bi_LSTM_3LAYERS	PReLU
GRU_1LAYER	ReLU
GRU_2LAYERS	ReLU
GRU_3LAYERS	Swish

5.3.4 Neuron Number

Figure 9 shows the influences of different neuron number has on the performance of RUL prediction with a fixed optimizer (Adam), activation function (ReLU) and sequence length (50). Gradient exploring, or gradient vanishing occurs in this case when using network structure RNN_1LAYER and all GRU structures which may suggest that neuron number is a sensitive parameter for RNN and GRU in terms of stability. The performance of different neuron number varies significantly using different algorithms. Taking the LSTM network structure as an example, the influence of different neuron numbers is smaller when using LSTM_1LAYER than the other two. In addition, for LSTM_3LAYERS, the observation shows that the more neuron number is used, the less accurate the result turns out to be, while this tendency cannot be found in the other two network structures. As only three different neuron numbers were tested, the optimal neuron for each network structure could not be achieved. Nevertheless, the optimal neuron number for different algorithms in this case is listed in Table 6 just for reference.

5.3.5 Overall Performance of Each Algorithm

Figure 10 demonstrates the performance of different algorithms using a certain group of parameters. As in this case, gradient exploring, or gradient vanishing occurred when using RNN_1LAYER and GRU_3LAYRES. It seems that generally, the performance of LSTM, Bi_LSTM and GRU network structures seems to be relatively close and significantly better than RNN. The accuracy of LSTM is close to Bi_LSTM and GRU, but the precision is relatively poor. GRU turns out to be very accurate and precise, but it suffers from stability problems. Thus, a Bi_LSTM structure might be a better option in this case.

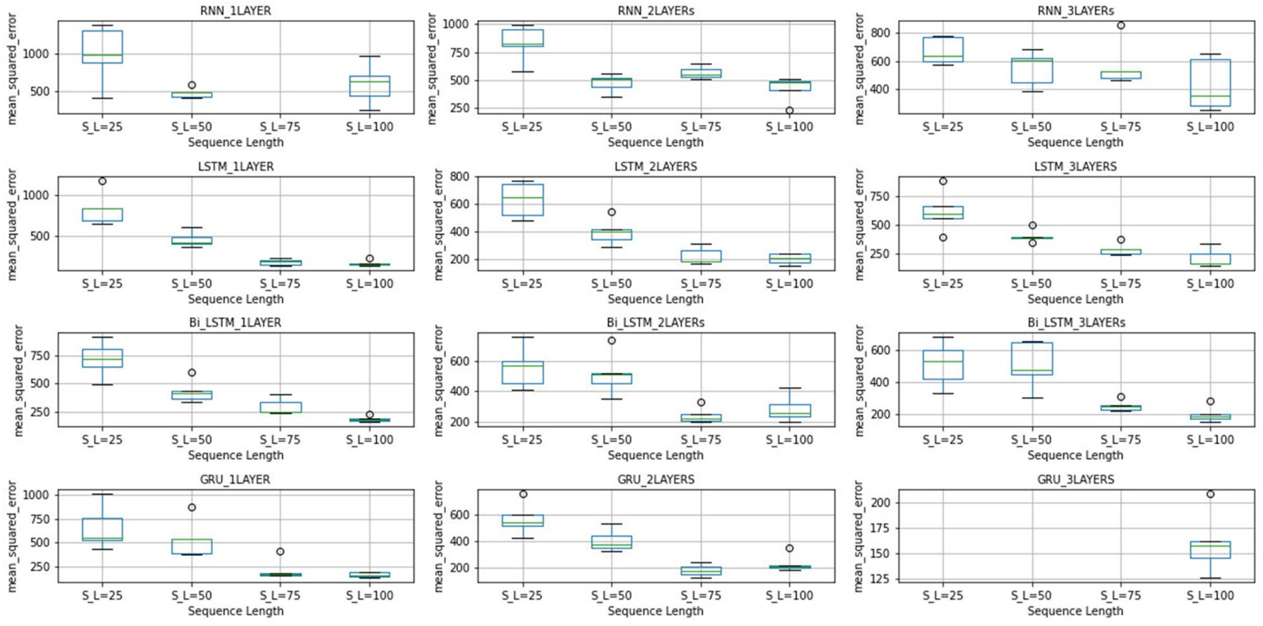

Figure 8 Performance of different sequence length on the test dataset

Table 5 Optimal sequence length for different algorithms

Algorithms	Optimal sequence length
RNN_1LAYER	50
RNN_2LAYERS	50,75,100
RNN_3LAYERS	75
LSTM_1LAYER	100
LSTM_2LAYERS	100
LSTM_3LAYERS	100
Bi_LSTM_1LAYER	100
Bi_LSTM_2LAYERS	75
Bi_LSTM_3LAYERS	100
GRU_1LAYER	75,100
GRU_2LAYERS	100
GRU_3LAYERS	100

Table 6 Optimal Neuron number for different algorithms

Algorithms	Optimal Neuron number
RNN_1LAYER	64,32
RNN_2LAYERS	32
RNN_3LAYERS	32
LSTM_1LAYER	128
LSTM_2LAYERS	32
LSTM_3LAYERS	32
Bi_LSTM_1LAYER	32
Bi_LSTM_2LAYERS	64
Bi_LSTM_3LAYERS	64
GRU_1LAYER	128
GRU_2LAYERS	128
GRU_3LAYERS	64

A more detailed comparison of different algorithms is displayed in Table 7. The optimal parameters (with the base parameters) for this subset are highlighted using a yellow hatch for every network structure. Although the global optimal parameters cannot be selected for the dataset based on this table since it has not considered all combinations, it provides a level of useful options with certainty.

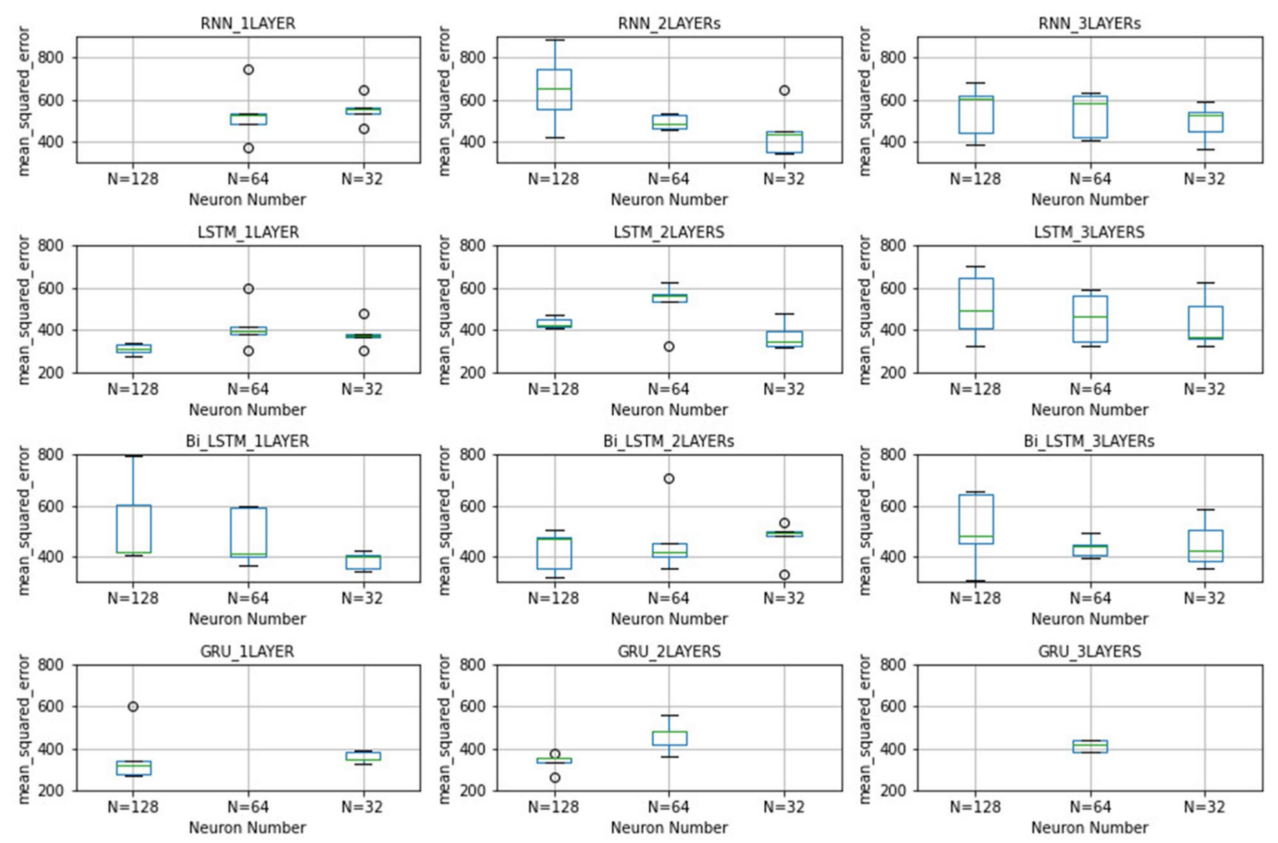

Figure 9 Performance of different Neuron number on test dataset

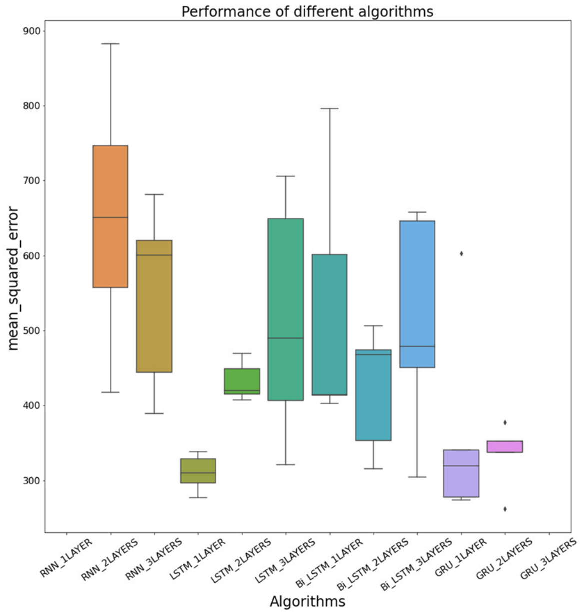

Figure 10 Performance of different algorithms on Dataset FD001, where Optimizer: Adam; Activation function: ReLU; Neuron Number: 128; Sequence Length: 50

6 Conclusions

A systematic review of the applications of RNN and its variants for RUL prediction in recent years is presented in this paper. An evaluation of these algorithms has been conducted using the NASA's C-MAPSS dataset, where different parameters, such as optimisers, activation functions, neuron quantities, and sequence length are discussed using a sensitivity analysis. It can be seen that some of the network structures are very parameter sensitive from the result of the evaluation. The influence of these parameters on the performance of RUL prediction is different according to different network structures. Instead of giving the optimal parameters and the network structures for the dataset, the result of this case study offers some practical choices of parameters for different network structures. Although the conclusions achieved above from this case study could not be applied to other cases directly, it at least suggests the influence of these factors on the RUL prediction. Moreover, it provides some options for researchers when they consider adopting DL to carry out the similar prediction task for their own cases.

Table 7 The average MSE recorded with regards to different parameters

	Optimizer				Activation function					Neuron Number		Sequence Length				
Algorithms	**ReLU**	**L_ReLU**	**PReLU**	**ELU**	**Swish**	**Rmsprop**	**Adam**	**AdaGrad**	**AdaDelta**	**128**	**64**	**32**	**25**	**50**	**75**	**100**
RNN_1L		569			871	569	487	286	478		533	553	994	478		596
RNN_2L	651	491	710	488	646	494	406		476	651	494	444	930	476	565	425
RNN_3L	547	563	539	440	831	325	572		547	547	531	493	670	547	570	431
LSTM_1L	310	443	439	382	725	566	397	260	464	310	419	379	836	464	196	181
LSTM_2L	432	526	391	464	670	398	358		396	432	522	369	632	396	221	199
LSTM_3L	515	460	480	531	872	411	375		397	515	458	436	617	397	284	209
Bi_LSTM_1L	526	458	423	516	562	718	422	362	428	526	473	385	714	428	294	185
Bi_LSTM_2L	423	431	538	507	672	441	413	228	513	423	465	467	557	513	243	286
Bi_LSTM_3L	508	492	494	611	673	473	434	285	508	508	435	450	512	508	253	198
GRU_1L	363	393	356	392	450	457	383	389	540	363		360	658	540	220	166
GRU_2L	336	415	383	423	486	417	365	535	405	336	459		570	405	179	231
GRU_3L		458			410	417	398				414					160

Appendix 1. Performance evaluation—Optimizers

Recommended optimizers for different structures.

RNN-1LAYER

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	4	4	
Adam	1	3	3	
AdaGrad	1	1	1	Recommended
AdaDelta	1	2	2	

RNN-2LAYERS

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	3	2	
Adam	1	1	1	Recommended
AdaGrad	0	0	0	
AdaDelta	1	2	3	

RNN-3LAYERS

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	1	1	Recommended
Adam	1	2	2	
AdaGrad	0			
AdaDelta	0			

LSTM-1LAYER

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	4	4	
Adam	1	1	2	
AdaGrad	1	2	1	Recommended
AdaDelta	1	3	3	

LSTM-2LAYERS

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	3	3	
Adam	1	1	1	Recommended
AdaGrad	0	0	0	
AdaDelta	1	2	2	

LSTM-3LAYERS

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	3	3	
Adam	1	2	1	Recommended
AdaGrad	0	0	0	
AdaDelta	1	1	2	Recommended

Bi_LSTM-1LAYER

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	4	4	
Adam	1	1	3	Recommended
AdaGrad	1	3	1	Recommended
AdaDelta	1	2	2	Recommended

Bi_LSTM-2LAYERS

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	2	2	
Adam	1	4	3	
AdaGrad	1	1	1	Recommended
AdaDelta	1	3	4	

Bi_LSTM-3LAYERS

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	3	3	
Adam	1	1	2	
AdaGrad	1	2	1	Recommended
AdaDelta	0	0	0	

GRU_1LAYER

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	3	3	
Adam	1	1	2	Recommended
AdaGrad	1	2	1	Recommended
AdaDelta	1	4	4	

GRU_2LAYERS

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	2	2	
Adam	1	1	1	Recommended
AdaGrad	1	3	4	
AdaDelta	1	4	3	

GRU_3LAYERS

Optimizers	Workability	Precision	Accuracy	Assessment
Rmsprop	1	2	2	
Adam	1	1	1	Recommended
AdaGrad	0	0	0	
AdaDelta	0	0	0	

2. Performance evaluation—Activation functions

Recommended activation functions for different structures.

RNN-1LAYER

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	0	0	0	
Leaky_ReLU	1	1	1	Recommended
PReLU	0	0	0	
ELU	0	0	0	
Swish	1	2	2	

RNN-2LAYERS

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	1	5	5	
Leaky_ReLU	1	1	1	Recommended
PReLU	1	4	4	
ELU	1	3	2	Recommended
Swish	1	2	3	

RNN-3LAYERS

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	1	4	3	
Leaky_ReLU	1	2	4	
PReLU	1	3	2	
ELU	1	1	1	Recommended
Swish	1	5	5	

LSTM-1LAYER

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	1	2	1	Recommended
Leaky_ReLU	1	4	3	
PReLU	1	1	2	
ELU	1	3	4	
Swish	1	5	5	

LSTM-2LAYERS

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	1	1	2	
Leaky_ReLU	1	3	4	
PReLU	1	2	1	Recommended
ELU	1	5	3	
Swish	1	4	5	

LSTM-3LAYERS

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	1	4	4	
Leaky_ReLU	1	3	2	
PReLU	1	2	3	
ELU	1	1	1	Recommended
Swish	1	5	5	

Bi_LSTM-1LAYER

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	1	4	3	
Leaky_ReLU	1	3	2	
PReLU	1	2	1	Recommended
ELU	1	1	4	
Swish	1	5	5	

Bi_LSTM-2LAYERS

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	1	2	1	Recommended
Leaky_ReLU	1	1	2	Recommended
PReLU	1	4	4	
ELU	1	3	3	
Swish	1	5	5	

Bi_LSTM-3LAYERS

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	1	3	3	
Leaky_ReLU	1	4	1	
PReLU	1	1	2	Recommended
ELU	1	5	5	
Swish	1	2	4	

GRU_1LAYER

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	1	1	1	Recommended
Leaky_ReLU	1	4	3	
PReLU	1	2	2	
ELU	1	5	4	
Swish	1	3	5	

GRU_2LAYERS

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	1	1	1	Recommended
Leaky_ReLU	1	4	3	
PReLU	1	3	2	
ELU	1	5	5	
Swish	1	2	4	

GRU_3LAYERS

Activation functions	Workability	Precision	Accuracy	Assessment
ReLU	0	0	0	
Leaky_ReLU	1	1	2	
PReLU	0	0	0	
ELU	0	0	0	
Swish	1	2	1	Recommended

3. Performance evaluation—Sequence length

Recommended sequence length for different structures.

RNN-1LAYER

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	3	3	
50	1	1	1	Recommended
75	0	0	0	
100	1	2	2	

RNN-2LAYERS

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	4	4	
50	1	2	2	Recommended
75	1	1	3	Recommended
100	1	3	1	Recommended

RNN-3LAYERS

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	2	4	
50	1	3	3	
75	1	1	2	Recommended
100	1	4	1	

LSTM-1LAYER

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	4	4	
50	1	3	3	
75	1	2	2	
100	1	1	1	Recommended

LSTM-2LAYERS

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	4	4	
50	1	3	3	
75	1	2	2	
100	1	1	1	Recommended

LSTM-3LAYERS

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	4	4	
50	1	1	3	
75	1	2	2	
100	1	3	1	Recommended

Bi_LSTM-1LAYER

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	4	4	
50	1	3	3	
75	1	2	2	
100	1	1	1	Recommended

Bi_LSTM-2LAYERS

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	4	4	
50	1	2	3	
75	1	1	1	Recommended
100	1	3	2	

Bi_LSTM-3LAYERS

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	4	3	
50	1	3	4	
75	1	1	2	
100	1	2	1	Recommended

GRU_1LAYER

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	4	4	
50	1	3	3	
75	1	1	2	Recommended
100	1	2	1	Recommended

GRU_2LAYERS

Sequence length	Workability	Precision	Accuracy	Assessment
25	1	4	4	
50	1	3	3	
75	1	2	2	
100	1	1	1	Recommended

GRU_3LAYERS

Sequence length	Workability	Precision	Accuracy	Assessment
25	0	0	0	
50	0	0	0	
75	0	0	0	
100	1	1	1	Recommended

4. Performance evaluation—Neuron number

Recommended Neuron number for different structures.

RNN-1LAYER

Neuron number	Workability	Precision	Accuracy	Assessment
128	0	0	0	
64	1	2	1	Recommended
32	1	1	2	Recommended

RNN-2LAYERS

Neuron number	Workability	Precision	Accuracy	Assessment
128	1	3	3	
64	1	1	2	
32	1	2	1	Recommended

RNN-3LAYERS

Neuron number	Workability	Precision	Accuracy	Assessment
128	1	2	3	
64	1	3	2	
32	1	1	1	Recommended

LSTM-1LAYER

Neuron number	Workability	Precision	Accuracy	Assessment
128	1	3	1	Recommended
64	1	2	3	
32	1	1	2	

LSTM-2LAYERS

Neuron number	Workability	Precision	Accuracy	Assessment
128	1	1	2	
64	1	2	3	
32	1	3	1	Recommended

LSTM-3LAYERS

Neuron number	Workability	Precision	Accuracy	Assessment
128	1	3	3	
64	1	2	2	
32	1	1	1	Recommended

Bi_LSTM-1LAYER

Neuron number	Workability	Precision	Accuracy	Assessment
128	1	3	3	
64	1	2	2	
32	1	1	1	Recommended

Bi_LSTM-2LAYERS

Neuron number	Workability	Precision	Accuracy	Assessment
128	1	3	2	
64	1	2	1	Recommended
32	1	1	3	

Bi_LSTM-3LAYERS

Neuron number	Workability	Precision	Accuracy	Assessment
128	1	3	3	
64	1	1	1	Recommended
32	1	2	2	

GRU_1LAYER

Neuron number	Workability	Precision	Accuracy	Assessment
128	1	2	1	Recommended
64	0	0	0	
32	1	1	2	

GRU_2LAYERS

Neuron number	Workability	Precision	Accuracy	Assessment
128	1	1	1	Recommended
64	1	2	2	
32	0	0	0	

GRU_3LAYERS

Neuron number	Workability	Precision	Accuracy	Assessment
128	0	0	0	
64	1	1	1	Recommended
32	0	0	0	

Acknowledgements
Not applicable.

Authors' Information
Youdao Wang, born in 1992, is currently a PhD candidate at *Cranfield University, UK*. He received his Msc degree from *The University of Manchester, UK*, in 2017. His research interests focus on adopting DL algorithms into the RUL prediction of industrial systems.

Yifan Zhao was born in Zhejiang, China. He received the PhD degree in Automatic Control and System Engineering from *the University of Sheffield, UK*, in 2007. He is currently a Senior Lecturer in Data Science at *Cranfield University, UK*. His research interests include computer vision, signal processing, non-destructive testing, active thermography, and nonlinear system identification.

Sri Addepalli is a Lecturer in Degradation Assessment at *Cranfield University, UK*. His research interests include passive and active thermography and material component damage characterisation with specific expertise in composite materials. Sri holds an MPhil in Materials Technology from *Swansea University, UK* and a BEng in Mechanical Engineering from *Anna University, India*. He is also a member of the Institute of Engineering Technology.

Authors' Contributions
YZ was in charge of the whole trial; YW wrote the manuscript; SA assisted with editing. All authors read and approved the final manuscript.

Funding
Supported by U.K. EPSRC Platform Grant (Grant No. EP/P027121/1).

Competing Interests
The authors declare no competing financial interests.

Received: 6 October 2020 Revised: 17 June 2021 Accepted: 22 June 2021
Published online: 12 July 2021

References

[1] P E Leser. *Probabilistic prognostics and health management for fatigue-critical components using high-fidelity models*. North Carolina State University, ProQuest Dissertations Publishing, 2017. 10758909
[2] T Salunkhe, N I Jamadar, S B Kivade. Prediction of remaining useful life of mechanical components-A review. *International Journal of Engineering Education*, 2014, 3(6): 125-135.
[3] C Okoh, R Roy, J Mehnen, et al. Overview of remaining useful life prediction techniques in through-life engineering services. *Procedia CIRP*, 2014, 16: 158-163, https://doi.org/10.1016/j.procir.2014.02.006.
[4] J Ma, H Su, W Zhao, et al. Predicting the remaining useful life of an aircraft engine using a stacked sparse autoencoder with multilayer self-learning. *Complexity*, 2018: 1-13, https://doi.org/10.1155/2018/3813029.
[5] J Deutsch, D He. Using deep learning-based approach to predict remaining useful life of rotating components. *Journal of Zoo and Wildlife Medicine*, 2017, 40(2): 321-327, https://doi.org/10.1109/TSMC.2017.2697842.
[6] W Zhang, M P Jia, L Zhu, et al. Comprehensive overview on computational intelligence techniques for machinery condition monitoring and fault diagnosis. *Chinese Journal of Mechanical Engineering*, 2017, 30(4): 782-795, https://doi.org/10.1007/s10033-017-0150-0.
[7] A Kabir, C Bailey, H Lu, et al. A review of data-driven prognostics in power electronics. 35th International Spring Seminar on Electronics Technology, 2012: 189–192, https://doi.org/10.1109/ISSE.2012.6273136.
[8] R Zhao, R Yan, Z Chen, et al. Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, 2019, 115: 213-237, https://doi.org/10.1016/j.ymssp.2018.05.050.
[9] M Kraus, S Feuerriegel. Forecasting remaining useful life: Interpretable deep learning approach via variational Bayesian inferences. *Decision Support Systems*, 2019, 125, https://doi.org/10.1016/j.dss.2019.113100.
[10] R Zhao, R Yan, Z Chen, et al. Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, 2019, 115: 213–237.
[11] F Jia, Y Lei, J Lin, et al. Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data. *Mechanical Systems and Signal Processing*, 2016, 72-73: 303-315, https://doi.org/10.1016/j.ymssp.2015.10.025.
[12] P Wang, Ananya, R Yan, et al. Virtualization and deep recognition for system fault classification. *Journal of Manufacturing Systems*, 2017, 44: 310–316, https://doi.org/10.1016/j.jmsy.2017.04.012.
[13] H Shao, H Jiang, H Zhao, et al. A novel deep autoencoder feature learning method for rotating machinery fault diagnosis. *Mechanical Systems and Signal Processing*, 2017, 95: 187-204, https://doi.org/10.1016/j.ymssp.2017.03.034.
[14] L Liao, W Jin, R Pavel. Enhanced restricted Boltzmann machine with prognosability regularization for prognostics and health assessment. *IEEE Transactions on Industrial Electronics*, 2016, 63(11): 7076–7083, https://doi.org/10.1109/TIE.2016.2586442.
[15] M Gan, C Wang, C Zhu. Construction of hierarchical diagnosis network based on deep learning and its application in the fault pattern recognition of rolling element bearings. *Mechanical Systems and Signal Processing*, 2016, 72–73: 92–104, https://doi.org/10.1016/j.ymssp.2015.11.014.

[16] J Wang, Y Ma, L Zhang, et al. Deep learning for smart manufacturing: Methods and applications. *Journal of Manufacturing Systems*, 2018, 48: 144-156, https://doi.org/10.1016/j.jmsy.2018.01.003.

[17] S Zheng, K Ristovski, A Farahat, et al. Long short-term memory network for remaining useful life estimation. *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, 2017: 88–95, https://doi.org/10.1109/ICPHM.2017.7998311.

[18] S Hochreiter, J Schmidhuber. Long short-term memory. *Neural Computation*, 1997, 9(8): 1735-1780, https://doi.org/10.1162/neco.1997.9.8.1735.

[19] M Yuan, Y Wu, L Lin. Fault diagnosis and remaining useful life estimation of aero engine using LSTM neural network. *2016 IEEE International Conference on Aircraft Utility Systems (AUS)*, 2016: 135–140, https://doi.org/10.1109/AUS.2016.7748035.

[20] M Yuan, Y Wu, S Dong, et al. Remaining useful life estimation of engineered systems using vanilla LSTM neural networks. *Neurocomputing*, 2018, 275: 167-179, https://doi.org/10.1016/j.neucom.2017.05.063.

[21] K Cho, B van Merrienboer, C Gulcehre et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014: 1724–1734, https://doi.org/10.3115/v1/d14-1179.

[22] J Chen, H Jing, Y Chang, et al. Gated recurrent unit based recurrent neural network for remaining useful life prediction of nonlinear deterioration process. *Reliability Engineering and System Safety*, 2019, 185: 372-382, https://doi.org/10.1016/j.ress.2019.01.006.

[23] R Zhao, R Yan, J Wang, et al. Learning to monitor machine health with convolutional Bi-directional LSTM networks. *Sensors (Switzerland)*, 2017, 17(2): 1-18, https://doi.org/10.3390/s17020273.

[24] J Zhang, P Wang, R Yan, et al. Long short-term memory for machine remaining life prediction. *Journal of Manufacturing Systems*, 2018, 48: 78-86, https://doi.org/10.1016/j.jmsy.2018.05.011.

[25] Ahmed Elsheikh, Soumaya Yacout, Mohamed-Salah Ouali, et al. Bidirectional handshaking LSTM for remaining useful. *Neurocomputing*, 2019, 323: 148-156

[26] S Ruder. An overview of gradient descent optimization algorithms. 2016: http://arxiv.org/abs/1609.04747.

[27] D P Kingma, J L Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, San Diego, 2015: 1–15.

[28] P Ramachandran, B Zoph, Q V Le. Searching for activation functions. *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*, 2018: 1–13.

[29] A Saxena, K Goebel, D Simon, et al. Damage propagation modeling for aircraft engine run-to-failure simulation. *2008 International Conference on Prognostics and Health Management, PHM 2008*, 2008, https://doi.org/10.1109/PHM.2008.4711414.