Time Series Reconstruction using a Bidirectional Recurrent Neural Network based Encoder-Decoder Scheme

Wennian Yu[1], II Yong Kim[1], Chris Mechefske[1]

[1]*Department of Mechanical and Materials Engineering, Queen's University, Kingston, ON, Canada, K7L 3N6*

Abstract

In this paper, we introduce a bidirectional recurrent neural network (RNN) based encoder-decoder scheme to learn efficient and robust embeddings for a high-dimensional multivariate time series. The learnt embeddings can be regarded as the representations of input time series in a low dimensional space and can be used to reconstruct the input time series. Two popular advanced RNN architectures were employed to construct the proposed data reconstruction scheme. A publicly available turbofan engine degradation dataset was analyzed. It was found that the bidirectional recurrent neural networks show improved reconstruction performance compared with their unidirectional counterparts. Further parameter studies were conducted to investigate the effects of several key parameters on the reconstruction performance in terms of the reconstruction error and the training time.

Keywords: time series; bidirectional recurrent neural network; encoder-decoder; embeddings

Introduction

With the continuous development of the instrumentation technology and computation system, many companies can afford to continuously collecting data from operating machines via numerous types of sensors for the routinely monitoring of the machine health. There is no doubt that the era of big data is coming [1]. The sensory signals collected from a machine are normally multivariate time series (i.e. high-dimensional) which usually possess complex dynamic patterns and nonlinear characteristics. Dimensionality reduction techniques, which convert the high-dimensional data to low-dimensional embeddings (codes), are thus necessary since they can facilitate the visualization and storage of high-dimensional data [2].

Principal components analysis (PCA) is the primary tool to reduce the dimensions of the original multivariate data. It is basically an orthogonal linear transformation that maps the high-dimensional correlated variables into low-dimensional uncorrelated components. Thus, PCA can only remove the linear correlations within the original data [3]. Various nonlinear versions of PCA were reported in the literature, such as the kernel principle component analysis [4] and autoencoder [2]. In deep learning, a regular autoencoder is basically a multilayer artificial neural network (NN) used to learn efficient data coding in an unsupervised manner. It consists of a "encoder" network and a "decoder" network. The encoder transforms the high-dimensional data into low-dimensional codes, and the decoder reconstructs the original data based only on the codes. The autoencoder is

trained to recover the input in an unsupervised manner. It has been demonstrated that the PCA is a special case of autoencoder if linear activation functions of hidden layers are used.

For a multivariate time series, for instance, sensory data monitoring system degradation, the above-mentioned techniques may fail to capture the temporal patterns as they disregard the timing information in the time series. Cho et al. [5] introduced a novel recurrent neural network (RNN) based autoencoder (or encoder-decoder) for statistical machine translation. Compared with the feedforward NN, the RNN has an internal state (memory) which enable it to capture the temporal dependences within the time series. Based on this scheme, Malhotra et al. [6] proposed a long short-term memory based encoder-decoder (LSTM-ED) to obtain an unsupervised health index from the multi-sensor time-series data collected from a system. The LSTM-ED is trained to reconstruct the input time series. Later, Gugulothu et al. [7] proposed the gated recurrent unit based encoder-decoder (GRU-ED) to generate the low-dimensional embeddings from multivariate time series reflecting the health state of a system.

In this study, we present a bidirectional recurrent neural network based encoder-decoder (BiRNN-ED) scheme to learn efficient and robust embeddings for the input multivariate time series. The BiRNN-ED is trained to reconstruct the input time series in an unsupervised way. Compared with the standard unidirectional RNNs, the bidirectional RNNs process the time series in the forward and backward ways [8], enabling it to capture the time dependencies within the time series in the forward and backward manners. In this paper, this special structure of the bidirectional RNNs has been demonstrated to yield improved reconstruction performance compared with their unidirectional counterparts.

Methodology

Standard RNN

Figure 1(a) shows the architecture inside a hidden unit of a three-layer vanilla RNN (Elman network). The hidden neurons $\boldsymbol{H}_t$ receive inputs not just from their previous layer at the current time step ($\boldsymbol{I}_t$), but also from their outputs at the previous time step ($\boldsymbol{H}_{t\text{-}1}$). Mathematically, the information passing through the hidden layer can be expressed as:

$$\boldsymbol{H}_t = f(\boldsymbol{w}_x \boldsymbol{I}_t + \boldsymbol{w}_h \boldsymbol{H}_{t-1} + \boldsymbol{b}) \tag{1}$$

where $f(\cdot)$ is the activation function of the hidden layer. $\boldsymbol{w}_x$ is the weighting matrix connecting the inputs with the hidden layer, whereas $\boldsymbol{w}_h$ is the weighting matrix connecting the hidden layers of adjacent time steps. $\boldsymbol{b}$ is the bias vector of the hidden layer. However, the vanilla RNNs are only capable of handling time series with short-term dependencies due to the vanishing and exploding gradient problem when training vanilla RNNs. The Long short-term memory unit [9] and the gated recurrent unit [4] are the two popular building units for hidden layers of an RNN. They both adopt the gating mechanism to deal with the vanishing gradient problem.

A typical LSTM unit have three gates (the input gate, the forget gate, and the output gate) and a memory cell interacting with each other, as shown in Fig. 1(b). These gates and the cell are used to control the passing of information along the sequences, which can be mathematically expressed as:

$$
\begin{aligned}
\boldsymbol{i}_t &= \sigma(\boldsymbol{w}_{ix}\boldsymbol{I}_t + \boldsymbol{w}_{ih}\boldsymbol{H}_{t-1} + \boldsymbol{b}_i) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{w}_{ox}\boldsymbol{I}_t + \boldsymbol{w}_{oh}\boldsymbol{H}_{t-1} + \boldsymbol{b}_o) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{w}_{fx}\boldsymbol{I}_t + \boldsymbol{w}_{fh}\boldsymbol{H}_{t-1} + \boldsymbol{b}_f) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \otimes \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \otimes \phi(\boldsymbol{w}_{cx}\boldsymbol{I}_t + \boldsymbol{w}_{ch}\boldsymbol{H}_{t-1} + \boldsymbol{b}_c) \\
\boldsymbol{H}_t &= \boldsymbol{o}_t \otimes \phi(\boldsymbol{c}_t)
\end{aligned}
\tag{2}
$$

where $\boldsymbol{i}_t$, $\boldsymbol{o}_t$, $\boldsymbol{f}_t$ and $\boldsymbol{c}_t$ are the input gate, the output gate, the forget gate and the memory cell within units of the hidden layer respectively. $\boldsymbol{w}_{Gx}$ is the weighting matrix connecting the inputs with G (G = i, o, f and c representing the input gate, the output gate, the forget gate and the cell respectively). Similarly, $\boldsymbol{w}_{Gh}$ is the weighting matrix connecting the last hidden state $\boldsymbol{H}_{t\text{-}1}$ with G, and $\boldsymbol{b}_G$ is the corresponding bias vectors. $\sigma(\cdot)$ and $\phi(\cdot)$ are the sigmoid and the hyperbolic tangent functions respectively.

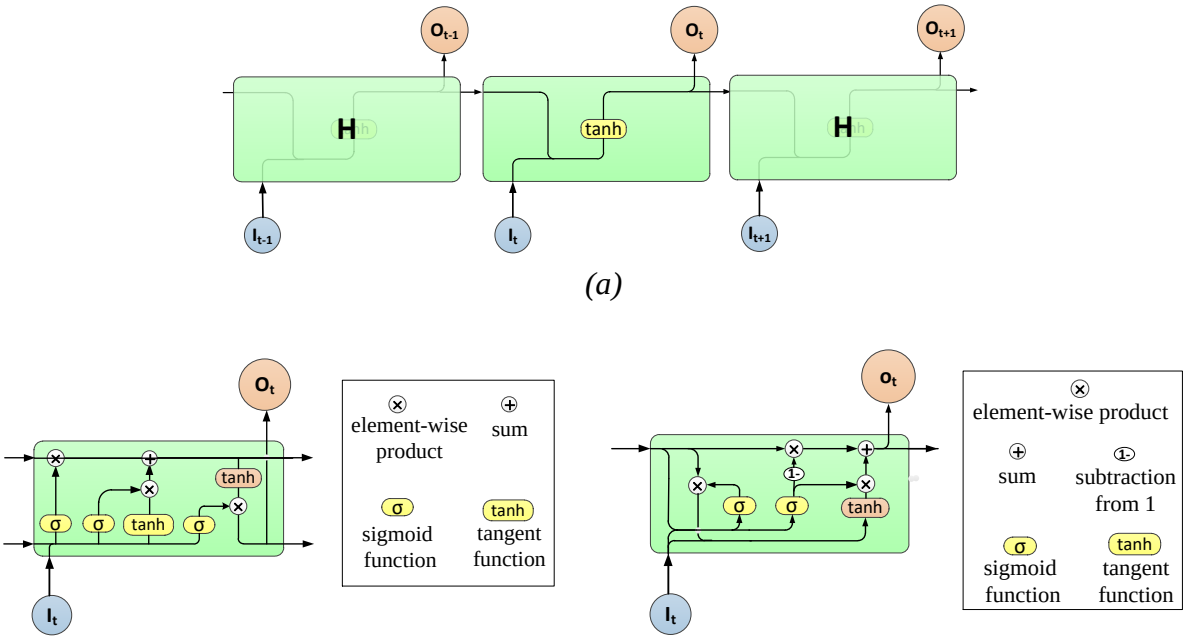

Figure 1: The architecture of a hidden neuron for: (a) Elman network, (b) LSTM [9], (c) GRU

Compared with the LSTM unit, a typical GRU unit is composed of only two gates (reset gate and update gate). Thus, it has fewer parameters than the LSTM unit. The architecture of the GRU based hidden layer can be mathematically expressed as:

$$
\begin{aligned}
\boldsymbol{r}_t &= \sigma(\boldsymbol{w}_{rx}\boldsymbol{I}_t + \boldsymbol{w}_{rh}\boldsymbol{H}_{t-1} + \boldsymbol{b}_r) \\
\boldsymbol{u}_t &= \sigma(\boldsymbol{w}_{ux}\boldsymbol{I}_t + \boldsymbol{w}_{uh}\boldsymbol{H}_{t-1} + \boldsymbol{b}_u) \\
\hat{\boldsymbol{H}}_t &= \phi(\boldsymbol{w}_{hx}\boldsymbol{I}_t + \boldsymbol{w}_{hh}(\boldsymbol{r}_t \otimes \boldsymbol{H}_{t-1}) + \boldsymbol{b}_h) \\
\boldsymbol{H}_t &= (1 - \boldsymbol{u}_t) \otimes \boldsymbol{H}_{t-1} + \boldsymbol{u}_t \otimes \hat{\boldsymbol{H}}_t
\end{aligned}
\tag{3}
$$

where $\boldsymbol{r}_t$ and $\boldsymbol{u}_t$ are the reset gate and the update gate within units of the hidden layer respectively. $\hat{\boldsymbol{H}}_t$ is the temporary hidden state, which will be used to determine the final hidden state $\boldsymbol{H}_t$. $\boldsymbol{w}_{Gx}$, $\boldsymbol{w}_{Gh}$ and $\boldsymbol{b}_G$ (G = r, u and h) are the corresponding weighting matrices and bias.

Bidirectional RNN

Compared with standard RNNs, bidirectional RNNs maintain two groups of hidden layers, one for input sequence in the positive time direction (forward states) just like the structure of the standard RNNs, and the other for input sequence in the negative time direction (backward states). These two groups of hidden layers do not interact with each other, and their outputs are merged and connected to the same output layer.

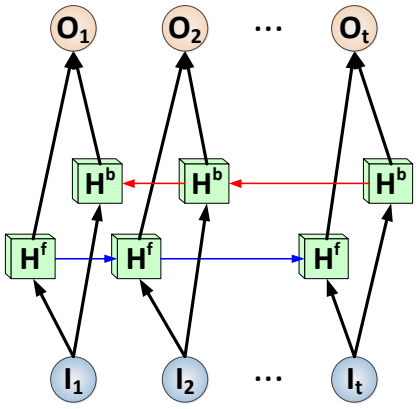

Figure 2: The structure of a bidirectional RNN

The mathematic expressions describing the architecture of bidirectional RNN are the same with that of the corresponding basic RNN, except there are two hidden states at the *t*th time step: $\boldsymbol{H}^f_t$ and $\boldsymbol{H}^b_t$ representing the hidden states resulting from the forward and backward processes respectively. The complete hidden representation $\boldsymbol{H}_t$ is the concatenation of the $\boldsymbol{H}^f_t$ and $\boldsymbol{H}^b_t$ as follows [8]:

$$\boldsymbol{H}_t = \boldsymbol{H}^f{}_t \oplus \boldsymbol{H}^b{}_t \tag{4}$$

Bidirectional RNN based Autoencoder

A regular autoencoder consists of an encoder and 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.

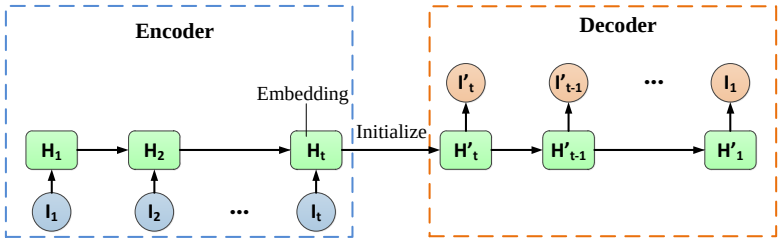

Figure 3: An RNN Encoder-Decoder [6, 7]

Figure 3 shows an RNN Encoder-Decoder network proposed by Malhotra et al. [6]. Given an input time sequence $\boldsymbol{\Omega} = \{\boldsymbol{I}_1, \boldsymbol{I}_2, \ldots, \boldsymbol{I}_t\}$, where each point $\boldsymbol{I}_i$ (i = 1, 2, …, t) is an m-dimensional vector depending on the dimension of input, which is either a univariate (i.e. m = 1) or a multivariate time

series. The encoder RNN iterates through each point $\boldsymbol{I}_i$ in the time sequence $\boldsymbol{\Omega}$ until the final point $\boldsymbol{I}_t$ so that the final hidden state $\boldsymbol{H}_t$ is obtained:

$$\boldsymbol{H}_t = f_e(\boldsymbol{\Omega}) \tag{5}$$

The decoder RNN has the same structure as the encoder RNN. However, it uses only the final hidden state of encoder $\boldsymbol{H}_t$ as the initial input to reconstruct the input time series in a reverse order $\boldsymbol{\Omega}' = \{\boldsymbol{I}'_t, \boldsymbol{I}'_{t-1}, \ldots, \boldsymbol{I}'_1\}$:

$$\boldsymbol{\Omega}' = f_d(\boldsymbol{H}_t) \tag{6}$$

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

$$E = \frac{1}{2}\sum_{i=1}^{t}(\|\boldsymbol{e}_i\|_2)^2 \tag{7}$$

where $\|\cdot\|_2$ is the 2-norm operator of a vector. Once the RNN Encoder-Decoder is well trained, the final hidden state of the encoder carries all the relevant information to reconstruct the input time series via the decoder. It is a usual practice to transform a multivariate time series into a one-dimensional vector representation via the autoencoder. Thus, if there are multiple hidden layers in the RNN autoencoder, the representation (embedding, or codding) $\boldsymbol{z}_t$ of the input is often obtained by the concatenation of the final hidden states from all the hidden layers in the encoder RNN:

$$\boldsymbol{z}_t = \boldsymbol{H}_t^1 \oplus \boldsymbol{H}_t^2 \oplus \ldots \boldsymbol{H}_t^L \tag{8}$$

where $\boldsymbol{H}_t^j$ is the final hidden state vector at the jth layer of encoder (j = 1, 2, … L), and L is the number of hidden layers.

Application and Results

Turbofan engine dataset

In this paper, we applied the methodology on the publicly available turbofan engine datasets *train_FD001.txt* provided by the NASA prognostic data repository [11]. It describes the health degradation of 100 engines (units) till a failure threshold is reached, i.e. run-to-failure multivariate time series. The dataset contains readings from 21 dependent sensors, which are contaminated with measurement noise. For dataset *train_FD001.txt*, it is characterized by one failure mode and one operating condition. Not all sensors are informative with the engine degradation. In this study, the 7 sensors as suggested by Wang et al. [3], which provide continuous-value readings and consistent trend (increasing or decreasing) with the degradation of the engines, are adopted for analysis. Their indexes are 2, 3, 4, 7, 11, 12, 15 respectively.

For the selected sensor sets, we use the z-score normalization to transform the sensor readings within acceptable range before inputting them into the autoencoder [6, 7]. Thus, the original time series $\boldsymbol{x}'_{m,k}$ corresponding to the mth sensor and kth operating condition are transformed by:

$$\boldsymbol{x}_{m,k} = \frac{\boldsymbol{x}'_{m,k} - \mu_{m,k}}{\sigma_{m,k}} \tag{9}$$

where $\mu_{m,k}$ and $\sigma_{m,k}$ are the mean value and the standard deviation for the mth sensor readings and kth operating condition over all instances. After normalization, a window with a fixed size W is

sliding along the sensor readings of each instance of the training set, as shown in Fig. 4. The resulting subsequences denoted by $\{\boldsymbol{\Omega}_1^{\{j\}}, \boldsymbol{\Omega}_2^{\{j\}}, \ldots, \boldsymbol{\Omega}_{Tj-W+1}^{\{j\}}\}$ for all instances available (j = 1, 2, …, J, and J is the number of instances or engines) are used to train the RNN based autoencoder. Once the RNN based autoencoder is trained, each point in the original sensor readings is predicted as many times as the number of the windows (subsequence) which include the point to be predicted [6]. Hence, the final prediction of a point is the average of its original predictions.

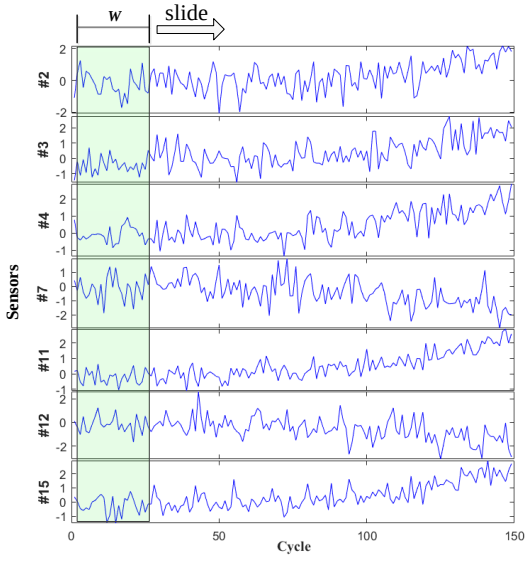

Figure 4: Sliding window on the normalized sensor readings of an instance

Parameter study

Without loss of generality, we randomly select 80 engines out of the 100 engines in *train_FD001.txt* to train the RNN based autoencoder models. The remaining 20 engines are used for the validation purpose. The effects of different parameters on the reconstruction performance of the RNN based autoencoder are studied in this section. The default values of some parameters are given in Table 1. The number of hidden layers is selected to be one, because it was found that multiple hidden layers didn't show obvious prediction improvement and the computation was significantly expensive. We didn't employ any regularization techniques (such as dropout) to enhance the robustness of the RNN based autoencoder, as the training samples are sufficient for the cased studied. The parameter study was carried out in a personal computer equipped with Inter Core i7 processor, 2.6 GHz clock speed and 16 GB RAM.

Table 1: Default values of some key parameters

Parameter	Value
Architecture	GRU / LSTM / **Bi-GRU** / Bi-LSTM
Number of hidden layers L	1
Number of hidden nodes N_h	20 / 50 / 100 / **200**
Window length W	5 / **10** / 20 / 30
Learning rate	0.02
Training epochs	2

Note: the bold values are the default values while conducting parameter study. For instance, the default architecture is Bi-GRU while varying the window length or the number of hidden nodes.

Table 2: Prediction performance using different parameters' values

Architecture				
	GRU	LSTM	Bi-GRU	Bi-LSTM
Validation error	0.6276	0.7179	0.4427	0.5946
Training time (s)	164.6	264.4	324.2	504.5
Number of hidden nodes N_h				
	20	50	100	200
Validation error	0.9091	0.7128	0.5732	0.4427
Training time (s)	51.09	71.03	145.4	324.2
Window length W				
	5	10	20	30
Validation error	0.1062	0.4427	0.8786	1.028
Training time (s)	268.2	324.2	423.2	524.8

Figure 5 show the original and reconstructed readings of a sensor from a unit in the validation set based on different RNN architectures while the other parameters are kept in the default values shown in Table 1. The validation errors (in terms of the root mean square of the difference between the original and reconstructed data) and the training times for different RNN architectures are listed in Table 2. Figure 6 and Fig. 7 show the similar comparisons using different numbers of hidden nodes and different window length respectively. The corresponding validation errors and the training times are listed in Table 2.

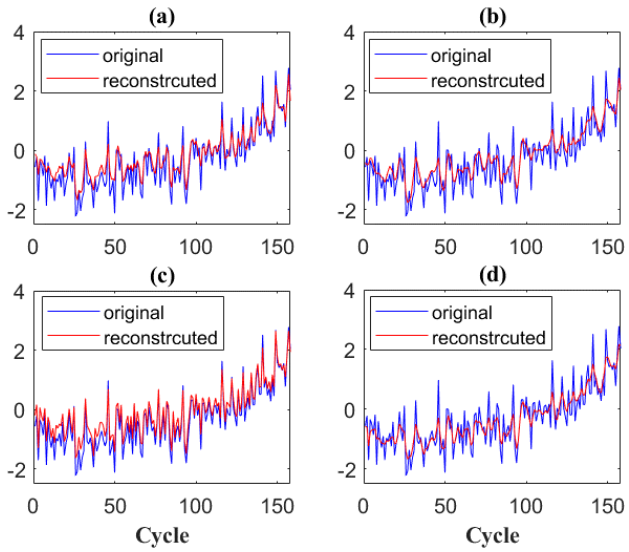

Figure 5: Comparisons between the original and reconstructed readings of a sensor from a unit in the validation set under different architectures: (a) GRU, (b) LSTM, (c) Bi-GRU, and (d) Bi-LSTM

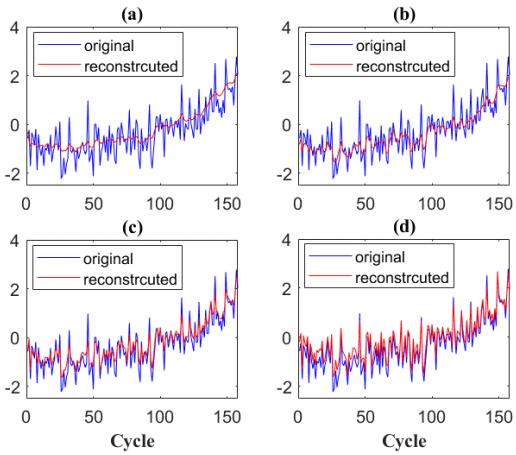

Figure 6: Comparisons between the original and reconstructed data of sensor readings from a unit in the validation set using different number of hidden nodes: (a) 20, (b) 50, (c) 100, and (d) 200

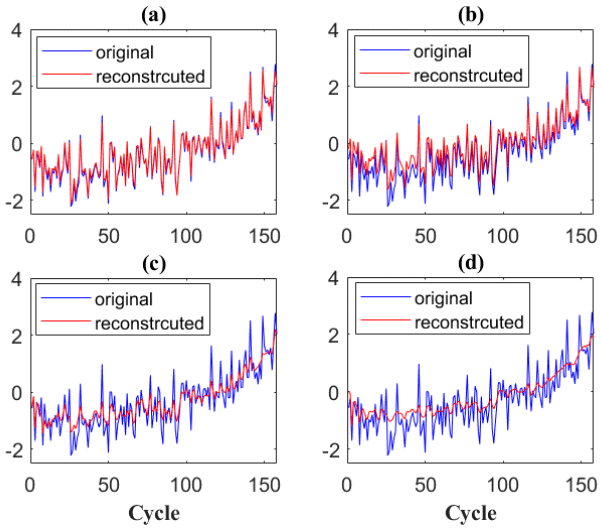

Figure 7: Comparisons between the original and reconstructed data of sensor readings from a unit in the validation set using different window lengths: (a) 5, (b) 10, (c) 20, and (d) 30

Discussions and Conclusions

Through the parameter studies performed in the last section, it can be found that the bidirectional RNNs show improved reconstruction performance compared with their unidirectional counterparts but with increased training time. This is due to the special structure of the bidirectional RNNs which was designed to increase the amount of input information to the neural networks. The GRU architecture is better than the LSTM architecture in terms of reconstruction error and training time for the case studied.

It is also noticed that with the number of the hidden nodes increases, the RNN based encoder-decoder scheme can more accurately reconstruct the input data but with proportionally increasing training time. On the other hand, fewer hidden nodes still ensure the RNN based autoencoder to

capture the underlying trend of the input as shown in Fig. 6(a). Thus, the RNN based autoencoder can be used to smooth the input time series and remove the noise if appropriately selecting the number of hidden nodes. In addition, it is found that that the smaller the window length *W*, the more accurate of the reconstruction, and the less of the training time. However, RNN based autoencoder with large window size can still capture the necessary pattern in the input time series and remove noise.

Even though an increasing number of hidden nodes and decreasing sliding window length will result in more accurate reconstruction of input time series, the size of the embeddings (used to recover the input) will also significantly increase. Thus, appropriate selections of these two parameters' values are required in order to achieve a reasonable balance between the reconstruction performance and the size of embeddings.

References

[1] Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., and Gao, R.X., "Deep Learning and Its Applications to Machine Health Monitoring: A Survey", 2016. Preprint arXiv:1612.07640. URL: https://arxiv.org/abs/1612.07640.

[2] Hinton, G.E. and Salakhutdinov, P.R., "Reducing the Dimensionality of Data with Neural Networks," Science, vol. 313, pp. 504–508, 2006.

[3] Wang, T., "Trajectory Similarity Based Prediction for Remaining Useful Life Estimation," Ph.D. Thesis. University of Cincinnati, Cincinnati, USA, 2010.

[4] Wang, J., Xie, J., Zhao, R., Zhang, L., and Duan, L., "Multisensory fusion based virtual tool wear sensing for ubiquitous manufacturing," Robot. Comput. Integr. Manuf., vol. 45, pp. 47–58, 2017.

[5] Cho, K., van Merrienboer B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," 2014. arXiv:1406.1078

[6] Malhotra, P., Tv, V., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., and Shroff G., "Multi-sensor prognostics using an unsupervised health index based on LSTM encoder-decoder", In 1st ACM SIGKDD Workshop on Machine Learning for Prognostics and Health Management, San Fransisco, CA, USA, 2016. URL: http://arxiv.org/abs/1607.00148.

[7] Gugulothu, N., Tv, V., Malhotra, P., Vig, L., Agarwal, P. and Shroff, G., "Predicting Remaining Useful Life using Time Series Embeddings based on Recurrent Neural Networks," CEUR Workshop Proc., vol. 1953, pp. 39–42, 2017. Preprint arXiv: 1709.01073, 2017.

[8] Zhao, R., Yan R., Wang, J. and Mao, K., "Learning to monitor machine health with convolutional Bi-directional LSTM networks," Sensors (Switzerland), vol. 17, no. 2, pp. 1–18, 2017.

[9] Olah, C., Understanding LSTM Networks. Posted on Web. Blog, 27 August 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed 13 March 2018.

[10] Hochreiter, S. and Schmidhuber, J., "Long short-term memory". Neural Comput., 9(8), 1735-1780, 1997.

[11] Saxena, A. and Goebel, K. (2008) "Turbofan Engine Degradation Simulation Data Set", NASA Ames Prognostics Data Repository (http://ti.arc.nasa.gov/project/prognostic-data-repository), NASA Ames Research Center, Moffett Field, CA