Research on Prognosis for Engines by LSTM Deep Learning Method

Liang Tang, Shunong Zhang, Xuesong Yang, Shuli Hu
School of Reliability and Systems Engineering
Beihang University
Beijing, China
Tangliang@buaa.edu.cn, zsn@buaa.edu.cn, yangxuesong@buaa.edu.cn, zy1714116@buaa.edu.cn

***Abstract*—Prognostics and health management (PHM) technology has been successfully applied in many complex equipment. However, with the equipment becoming more and more complex, the working conditions changing with time, and the equipment status information increasing, it is difficult by traditional technologies to cope with the new situation and new application scenarios. The application of deep learning method in many fields proves the ability of this method to deal with massive and complex data. In this paper, the special recurrent neural networks (RNN) called long-short term memory (LSTM) network are used to estimate the remaining life of engines with the data of PHM08 Challenge Competition. First, standardize the original data and add life labels in the data preprocessing stage. Then the influences of different data input methods on the prediction results are studied, and the results show that proper method is to input all the time series information at one time. The over-fitting phenomenon can be reduced to some extent by reducing the complexity of the neural network. Thus, a remaining life prediction method based on multi-dimensional data is obtained. The final result was uploaded to the competition's scoring system and got good results, which confirmed the accuracy of this method. Therefore, the article summarizes a highly accurate LSTM-based multidimensional data failure prediction method.**

Keywords—prognostics, deep learning, PHM, LSTM

I. INTRODUCTION

With the development of science and technology, modern industrial equipment and products are designed to perform more functions under dynamic conditions, resulting in more complex system components. During operation of a system eroded by external environment and internal wear or aging, some performances of the systems will gradually deviate from the preset ranges, and finally reach an unacceptable state: failure. For example, in an automobile production line, the overall economic loss caused by one-minute shutdown, or caused by mechanical failure or unqualified process control may be as high as several hundred thousand yuan [1]. Prognostics and health management (PHM) technology can provide solutions to this type of problem. The prognostics is to use the system state information obtained from various ways to evaluate the future state of the system, especially to predict the remaining useful life of the system, thereby supporting the health management. Since it is difficult to extract fault features from high-dimensional data, using deep learning artificial neural networks are used to automatically learn the implicit fault features, which may greatly reduce the knowledge needs of practitioners. It's unclear how a traditional neural network could use its reasoning about previous events to inform later ones. Recurrent neural networks (RNN) address this issue. They are networks with loops in them, allowing information to persist. The LSTM neural network is a special kind of recurrent neural network and performs well in time series prediction. This paper explore the use of LSTM deep learning methods to estimate the remaining useful life for engines with the data of PHM08 Challenge Competition because of its high data dimension that can represent the complexities of today's increasingly complex equipment. Moreover, PHM08 has a clear evaluation standard which can directly compare the pros and cons of the learning models.

II. RNN-LSTM DEEP LEARNING METHOD

A. RNN network

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feed forward neural networks, RNN can use their internal state (memory) to process sequences of inputs. Fig.1 shows the internal structure of RNN[2].

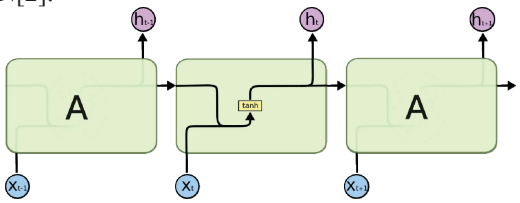


Figure 1. Internal structure of RNN

Fig.1 shows the situation in which RNN neurons (called memory cells) are expanded on a time axis. Each time there are two input values, two output values, and the hidden state is processed every iteration. Xt is input at time t, ht is the output at time t, And the things that each unit does are the same, that is ,RNN is an iterative use of a unit structure

978-1-7281-0860-5/19/$31.00 ©2019 IEEE

B. LSTM network

Long Short Term Memory network – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used[3].

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.

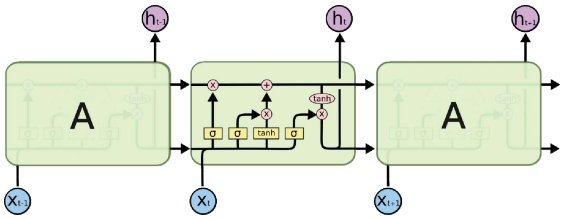


Figure 2. Internal structure of LSTM

All recurrent neural networks have the form of a repeating module chain of neural networks. In a standard RNN, the repeating module will have a very simple structure, such as a single tanh layer. LSTM also has this chain structure, but repeating modules have different structures. Compared to a simple layer of neural networks, LSTM has four layers. σ is an activation function, usually sigmoid. The first σ represents the forget gate, the second σ and tanh represents the input gate, the third σ represents the output gate.

III. DATA PREPROCESSING

A. Original Data

The original data from PHM08 Challenge Competition are provided with 26 columns of numbers, as shown in Table I. Each row is a snapshot of data taken during a single operational cycle; each column is a different variable. The columns correspond to:

1) unit number (U)
2) time, in cycles (T)
3) operational setting 1 (OP1)
4) operational setting 2 (OP2)
5) operational setting 3 (OP3)
6) sensor measurement 1 (S1)
7) sensor measurement 2 (S2)
...
26) sensor measurement 26 (S26)

TABLE I. THE STRUCTURE OF THE ORIGINAL DATA

U	T	OP1	OP2	OP3	S1	S2	S3	…
1	1	10.0047	0.2501	20	489.05	604.13	1499.45	…
1	2	0.0015	0.0003	100	518.67	642.13	1584.55	…
1	3	34.9986	0.8401	60	449.44	555.42	1368.17	…
1	4	20.0031	0.7005	0	491.19	607.03	1488.44	…
1	5	42.0041	0.8405	40	445.00	549.52	1354.48	…
1	6	20.0032	0.7017	0.0	491.19	607.37	1480.46	…
1	7	41.9998	0.84	40.0	445.0	549.57	1354.43	…
…	…	…	…	…	…	…	…	…

The original data contain training data sets and test data sets, which separately consist of multiple multivariate time series. Each time series is from a different engine – i.e., the data can be considered to be from a fleet of engines of the same type. Each engine starts with different degrees of initial wear and manufacturing variation which is unknown to the user. This wear and variation is considered normal, i.e., it is not considered a fault condition. There are three operational settings and 21 sensor measurements. The data are contaminated with sensor noise[6].

The engine is operating normally at the start of each time series, and starts to degrade at some point during the series. In the training set, the degradation grows in magnitude until a predefined threshold is reached beyond which it is not preferable to operate the engine. In the test set, the time series ends some time prior to complete degradation. The objective is to predict the number of remaining operational cycles before in the test set, i.e., the number of operational cycles after the last cycle that the engine will continue to operate properly.

B. Data Normalization

It can be seen from Table I that the scales of parameters are different. It should be separately performed normalization processing. In different characteristics, due to different dimensions or units, the order of magnitude of the data change interval is very different. If the data set is directly handed over to the neural network for processing, it may cause some small orders of magnitude parameters to be ignored or the influence to be weakened. If the parameters of small magnitude are important, problems will arise. Therefore the raw data should be normalized so that the parameters are on the same order of magnitude in order to synthesize the effects of the individual parameters.

Normalization is essentially a linear transformation of parameters that do not affect the relative state within the data. From the perspective of neural network training, if the data scales are very different, the error hyperplane of the parameter search will become steep, and the normalization will make the gradient change more gradual and more easy to converge.

There are many methods of normalization, such as linear normalization:

$$x' = \frac{\mathrm{x} - \min(x)}{\max(x) - \min(x)} \tag{1}$$

Z-score standardization, also known as standardization or standard deviation standardization:

$$\mathrm{x}^* = \frac{x - \mu}{\sigma} \tag{2}$$

The method of linear normalization is used herein, as shown in Table II.

TABLE II. DATA AFTER NORMALIZATION

U	T	OP1	OP2	OP3	S1	S2	S3	..
1	1	0.2382	0.297	0.2	0.5979	0.6295	0.6894	..
1	2	3.571e-05	3.563e-04	1	1	0.9789	0.9188	..
1	3	0.8331	0.9977	0.6	0.06027	0.1817	0.3354	..
1	4	0.4762	0.8319	0	0.627	0.6562	0.6597	..
1	5	0.9999	0.9982	0.4	0	0.1275	0.2985	..
1	6	0.4762	0.8334	0	0.627	0.6593	0.6382	..
1	7	0.9998	0.9976	0.4	0	0.128	0.2983	..
…	..	…	…	…	…	…	…	..

C. Add Life Label

Since the original data do not contain life information, it is necessary to manually generate a life indicator. The most intuitive way is, for each time point, to take the number of remaining time points from the end of life as the remaining life [7]. For example, sample #1 has 233 time series points, then take the numbers from 233 to 1 corresponding to the 233 time points as a label of life.

Since the number of time points of different samples varies from 128 to 356, for the long sequences, the pre-state is considered to be in good state, and the bound of good state is set to 130, i.e., the label of life greater than 130 is equal to 130, which is as the training objective of the neural network, as shown in Table III.

TABLE III. ADDING LIFE LABEL AND SET THE BOUND OF GOOD STATE AS 130

U	T	OP1	…	S1	…	Life Lable
1	1	0.2382	…	0.5979	…	130
1	2	3.571e-05	…	1.000	…	130
1	3	0.8331	…	0.06027	…	130
1	4	0.4762	…	0.6270	…	130
1	5	0.9999	…	0	…	130
1	6	0.4762	…	0.6270	…	130
1	7	0.9998	…	0	…	130
…	…	…	…	…	…	
1	222	0.4761	…	0.6270	…	2
1	223	0.8332	…	0.06027	…	1

IV. ESTIMATING REMAINING USEFUL LIFE BY LSTM

A. Running Environment

Benefiting from the development of deep learning frameworks such as Keras, Caffe, CNTK, Torch7, Theano, TensorFlow, etc., the construction of deep neural networks has become very convenient. Tensorflow is used as the backend, and the front end is Keras, because Keras is suitable for quickly trying various network structures in the exploration phase. More valuable is that Keras contains most of the latest deep learning methods and techniques.

Windows 10 operating system and Anaconda environment are used, Python version is Python 3.6. The computer hardware configuration is: i7-6700HQ, GTX970M, 16gbRAM.

B. Building LSTM Model

Under Keras, it is very convenient to set the network shape, activation function, loss function, add regular items, offset items, constraints and Dropout [8] and so on. The initialization method is a predefined Glorot [9] uniform distribution initialization method, also known as Xavier uniform initialization. Both regularization and Dropout are designed to enhance the generalization of the network and to avoid overfitting to some extent. Regularization reduces the model's weight update by constraining the weight-related parameters in the loss function, thus reducing the model's "point-by-point fitting" of the training set data. Dropout works in a simple way by randomly dropping a portion of the nodes in the network layer, using only the remaining nodes for calculations, and which nodes are lost in each iteration.

The optimizer generally is Sigmoid, tanh, ReLU, ADAM or RMSprop. ADAM and RMSprop are preferred in this study. The parameters of the two optimizers are slightly different. The most important parameters are learning rate and decay. The learning rate controls the speed at which the gradient decreases. Too small a learning rate can cause the training time to be too long, and an excessive learning rate may cause the model to linger near the optimal solution. Therefore, the addition of decay can make a large learning rate at the beginning of training, and as the training progresses, the learning rate is gradually reduced to avoid the situation near the optimal solution. In actual use, it is necessary to adjust the decay speed according to the number of iterations of training. The learning rate can also be adjusted by calling the underlying function. In the course of this research, the above methods have been used. After practice optimization and adjustment, the learning rate adjustment method of Fig. 3 is finally used.

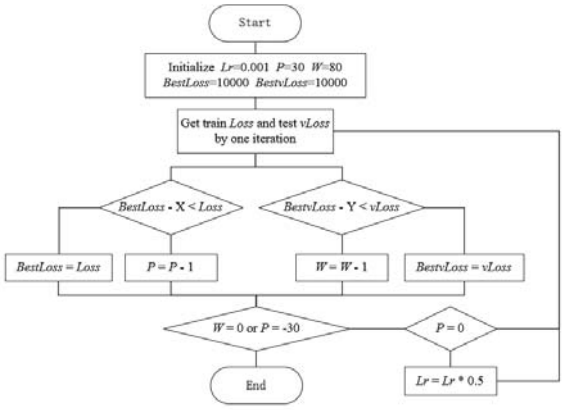


Figure 3. Adjustment Process of Learning Rate

This learning rate adjustment method needs to set three parameters: the initial learning rate Lr is 0.001; the patient parameter P=30, the larger the P, the slower the learning rate decreases, and the stop-over-fitting parameter W=80.

The best loss of the training set is recorded in Best-loss. The best loss of the testing set is recorded in Best-vloss. X and Y are the magnitudes of the loss reduction.

If the loss is not reduced to the best loss or P=0 after 30 iterations, the learning rate is halved, and the next iteration is entered, and P is reset to 30. The training will be stopped when P=-30 or testing set loss does not drop after W=80 times.

Simple neural networks are superimposed with network layers, and there are many parameters that can be adjusted for each layer. There is no direct explanation about the adaptability of the network, and it depends on continuous trying and correcting. In the field of deep learning, there have been many methods for automatically building networks in recent years, such as the automatic construction of RNNs in [10], but all require powerful computing power. The neural networks in this paper are all built by hand.

C. *Format of Input data*

The RNN in the deep learning framework Keras has special requirements for the data structure. The data format must be in the form of (sample, Timesteps, feature). To illustrate the data structure, the numbers of time series in Table IV represent the rows of the original data. For example, a sample with only one dimensional data, representing only one sensor collecting a string of data 1 to 9 over time, when timesteps is 1, the data is given to the model at a point in time in chronological order. When timesteps is 3, it is equivalent to resampling the data along the time axis with a window of size 3, and 3 time points are intercepted for each time and submitted to processing of the RNN learning model.

TABLE IV. TIMESTEPS EFFECT

Original Time	1	2	3	4	5	6	7	8	9
Timesteps = 1	1	2	3	4	5	6	7	8	9
Timesteps = 3	123	234	345	456	567	678	789	-	-

D. *Input Data in term of Time Series One by One*

The training data has 45918 rows and 26 columns, which is expressed to (45918, 26). The numbers of first column are from 1 to 218 representing 218 engine samples. The second column is time series. For the first engine sample 1#, there are 223 time points, meaning that the full life of the first engine sample 1# is 223, and the remaining 24 columns are the operational settings and sensor measurements. So the data format of the first engine sample 1# is (223, 24), and the data format of the second engine is (164, 24). The total life data onto a total of 218 engines is arranged downwards in order, with a total of 45918 lines.

Taking the data of the first engine as an example, the data is sequentially inputted in time order, and once input one time point, so format of the input data is (223, 1, 24). When the data input of the first engine is completed, input the data of the second engine by (164, 1, 24). A four-layer neural network is established. The first three layers are composed of LSTM nodes. The number of nodes in each layer is temporarily set to 48, 32, and 32 LSTM nodes according to experience. The fourth layer is responsible for integrated output by an LSTM node. As shown in Fig 12. The reason why four layers are chosen instead of five layers, six layers or higher is because more than four layers of LSTM cannot converge in test.

The total number of parameters of the neural network is 32840. The more parameters of the model, the stronger the ability of the model, and the wider the coverage of the model, the easier it is to include the solution of the problem. The solution space covered by the model with too few parameters is smaller, and the actual performance is that the model is under-fitting and does not converge. 32,840 parameters are sufficient for 45,918 data, usually the number of network parameters does not exceed the number of training samples[6].

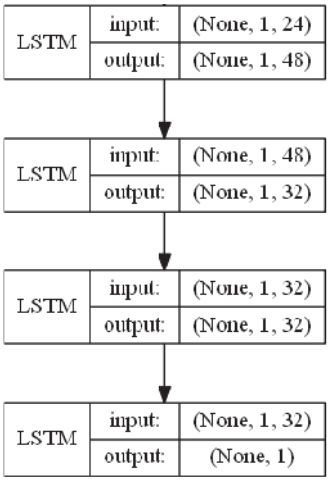


Figure 4. Model structure of input Data in term of time series

The neural network is trained on the training set. The training set is generated in such a way that the PHM08 data set train is divided into three equal parts, the first two are used as the training set, and the last one is used as the test set. The optimizer set as RMSprop, the initial learning rate is 0.01, and the error variation of the training process is shown in Fig. 5. In the figure, the horizontal axis represents epoch, the vertical axis represents MSE, the test MSE does not decrease at around 450, and the training ends at 298 epochs.

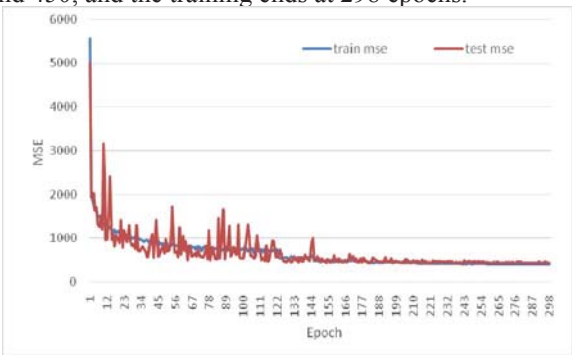
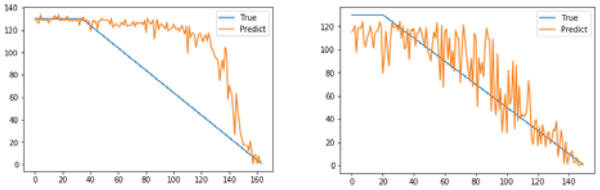
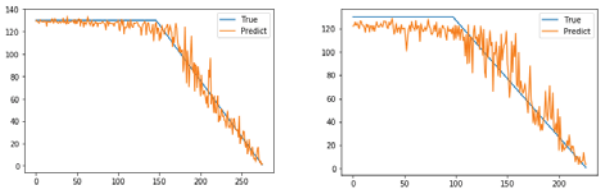


Figure 5. Model training process error variation



(a) 2# (b)8#



(c)111# (d)147#

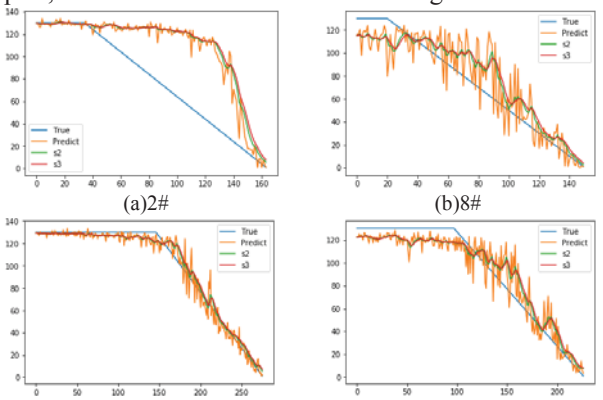
Figure 6. Model training process error changes

Using this model to predict several sample data, as shown in Fig. 6, the model shows that the model has better results of longer time series, such as #8 and #111, and the short time series is poorly fitted, such as # 2 and #3, the estimates for the end of life are more accurate in longer or shorter sequences. In general, however, the estimated life curve has significant oscillations, and the oscillations are more severe in shorter sequences, requiring smoothing.

The exponential smoothing method is chosen to solve the oscillating problem. The reason why the exponential smoothing is chosen is that the exponential smoothing only needs to know the data of the current time and the previous time to complete the smoothing, without acquiring the data of the whole curve or the latter moment. This feature fits well with the application scenario when estimating the remaining lifetime, because the estimated data at the next moment is not known in the actual applying scenario. The basic formula for exponential smoothing is as follows:

$$S_t = ay_t + (1-a)S_{t-1} \tag{3}$$

The result of exponential smoothing is shown in Fig. 7. It can be seen that after exponential smoothing, the errors of #3, #8, and #111 are slightly decreased, and the error of #2 is not lowered. The reason for the analysis is that the oscillation of #2 mainly occurs to one side of the real value, and the oscillations of #3, #8, and #111 are on both sides of the true value. The second exponential smoothing and the third exponential smoothing have different effects on different samples, but the overall difference is not large.



(a)2# (b)8#

(c)111# (d)147#

Figure 7. "Data in sequence" model index smoothing example

Network configurations like this had been tried, such as increasing the number of network layers, reducing the number of network layers, increasing the number of nodes per layer, reducing the number of nodes per layer, using different activation functions, using different optimizers, different Learning rate. Several of these configurations and errors are shown in Table V. There is no essential improvement in the results, and such results are not satisfactory.

TABLE V. PARTIAL NETWORK CONFIGURATION AND RESULTS

Network structure	Optimizer	Learning rate	Testing MES
L(24)-L(24)-L(24)-D(1)	ADAM	0.001	467.9
D(24)-D(12)-L(12)-L(12)-L(1)	ADAM	0.001	1063.7
L(12)-L(12)-D(1)	ADAM	0.001	518.7
L(12)-L(12)-L(1)	ADAM	0.001	526.7
L(32)-L(24)-L(1)	RMSprop	0.001	461.2
L(48)-L(32)-L(32)-L(1)	RMSprop	0.01	461.9
L(16)-L(16)-L(16)-L(16)-L(1)	RMSprop	0.001	471.2

Note: L = LSTM, D = Dense, the number of nodes in parentheses. For example, L(24) is a layer with 24 LSTM nodes.

Inputting data at a time point at one time may result in insufficient information received by the model, so we can consider using a fixed time window to input information for multiple consecutive time points at a time.

E. Data is Input in order of Times Window

The data input format of the previous section is (223, 1, 24), where the number 1 represents the window size, and this section will use different time windows. Using time windows to construct new data for raw data is essentially a resampling. This time window based resampling changes the data structure as a result of an increase in the amount of data at a single point in time and a reduction in total length of data. For example, using a time window with a window size of 5 to resample data of the form (223, 1, 24), the new data format generated is (219, 5, 24), if the time point is not allowed to be reused, the generated new data The format is (44, 5, 24). If resampling is allowed, only the last time life is estimated when the result is output. The input format is (219, 5, 24) and the output format is (219, 1, 1). If resampling is not allowed, the output is an estimate of the lifetime at each moment in the window, for example, the input format is (44, 5, 24) and the output format is (44, 5, 1). But the results of these two input methods are not ideal.

The network construction structure is shown in Fig. 8. It is a network with a data window size of 20. The data input format of the first sample is (11, 20, 24).

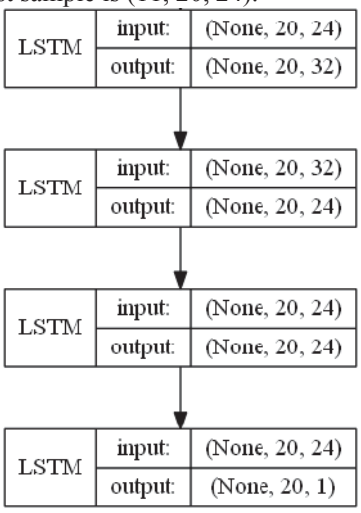


Figure 8. "Data input by time window" model structure

The network is trained on the training set, and the configuration of the training set is the same as previous section. The optimizer sets as RMSprop and the initial learning rate is 0.002. The error of the training process is shown in Fig. 17, the horizontal axis represents Epoch and the vertical axis represents MSE, the test MSE does not fall around 520, the training MSE reaches 360 or so. Finally, training ends at 133 epochs.

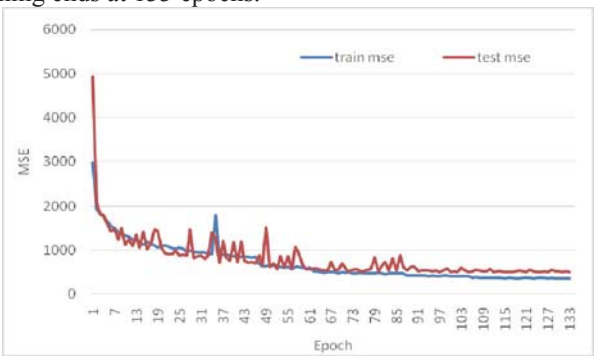
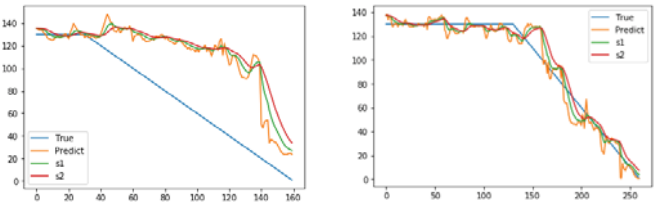


Figure 9."Data input by time window" model training process error variation

Using this model to predict the #2 and #8 sample data shown in Fig. 10, the smoothing process is the same as the previous section. It can be seen that the oscillating amplitude of the model is much less than that of the previous section without using the time window, but the performance of the test error is still not satisfactory.



(a)#2 (b)#8

Figure 10. "Data input by time window" model prediction example

A variety of attempts have been made to network configurations. Several of these configurations and errors are shown in Table VI. There is no essential improvement compared to the previous results, which is not satisfactory. After the previous section and the relevant experiments in this section, it is found that the type of activation function has little effect on the result. The optimizer RMSprop and ADAM have similar effect. The learning rate can be set to 0.001.

Although the test error is still large, the prediction curve is much smoother. This change provides an idea for further experiments. It is observed that the larger the time window is, the smoother the curve is. The next section will use the case where the time window is the largest, that is, inputting the entire data as a whole. Note 1: L = LSTM, D = Dense, the number of nodes in parentheses. For example, L(24) is a layer with 24 LSTM nodes.

TABLE VI. PARTIAL NETWORK CONFIGURATION AND RESULTS

structure	Time	Optimizer	Learning	Testing
	Window		rate	MSE
L(24)-L(24)-D(1)	3	ADAM	0.001	463.6
L(24)-L(24)-D(1)	5	ADAM	0.001	478.9
L(12)-L(12)-L(12)-D(1)	10	ADAM	0.001	465.1
L(24)-L(24)-L(12)-D(1)	3	ADAM	0.001	476.8

F. Input Method for Data Input as a Whole

Through the experiments in the first two sections, when the data is sequentially input into the network, the error can not continue to decrease at MSE=450 regardless of whether or not the time window is used for sampling. Note that using time window sampling makes the curve of the prediction result smoother, so try to input the data of each sample as a whole into the network.

Taking the data of the first engine sample 1# as an example, the overall input data structure is (1, 223, 24). Test network structure as used in Fig. 11, CuDNNLSTM in the picture is GPU- accelerated LSTM (still LSTM), can only run on GPU, running more than three times faster than ordinary LSTM. The network has a total of 33,185 parameters to be trained, and the activation function of the last layer is tanh:

$$\tan x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4}$$

The activation function is to add nonlinearity to the linear fully connected layer and enhance the nonlinearity of the model.

The artificial neural network was trained on the training set. The training set was slightly different from the previous configuration. The " train " was randomly divided into five parts, four for training and one for testing. Optimizer is set as RMSprop, initial learning rate is 0.001, an error of change in the training process could be seen in Figure 12, the horizontal axis represents epoch , the vertical axis represents loss = MSE , the test MSE does not fall at around 215 , and the training MSE maintains a falling trend of 112. The training ends in about Epoch. 206.

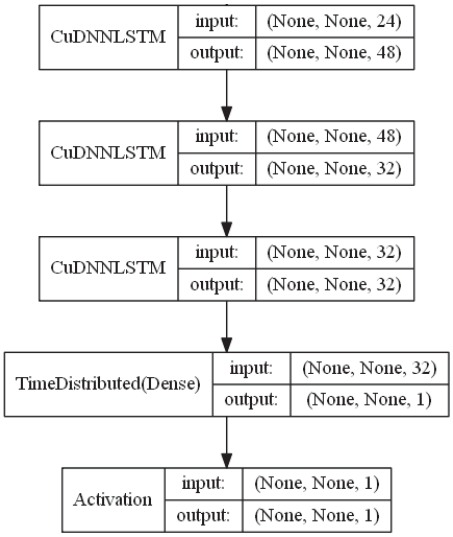


Figure 11. "Data overall input" model structure

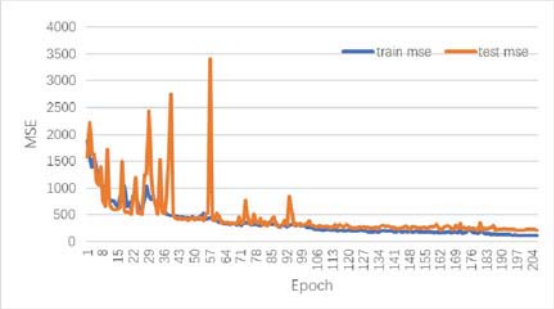
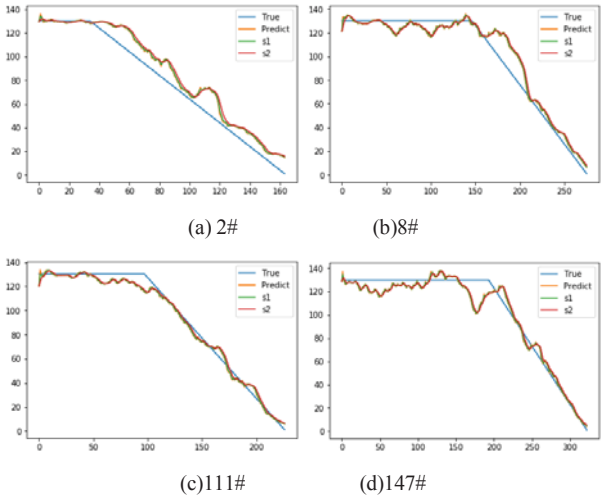


Figure 12. Error in the training process of the "data input" model

The presented model makes predictions for samples # 2, # 8, # 111, # 147 in Fig. 13.



(a) 2# (b)8# (c)111# (d)147#

Figure 13. "Data overall input" model prediction example

This result is compared with the best model results summarized in [7]. The error MSE has reached the first echelon, and the prediction results of the model on the test set are uploaded and directly compared with the results of other players. Create a new model and set the model equally. Retrain the entire" train" data set. After Epoch=170 , the model training MSE reaches 109.Then we could predict the test data, upload the outcome to the NASA data warehouse website, getting the score 28000 , This result is very bad. Obviously, the model is over-fitting because the training error is low and the high test error is the performance of over-fitting. The next section will address the overfitting problem.

V. SOLVING OVER-FITTING THROUGH SIMPLIFIED MODELS

Overfitting means that the model fits the training data too well, but it does not perform well on the test set, which is very common in deep learning. The more complex the model is, the stronger the model's ability to fit. The complexity of the model is reflected by the number of parameters. When there are enough parameters, the model can fit any existing data, including the noise in the data. Even if such a model fits well on the training data, it is not a good model, because the model does not have a useful insight into the data, just remember the training data recklessly. When the model encounters data that has not been seen, the performance is very bad. Especially for neural networks, there may be a large number of parameters. The model in the previous section contains 33,195 parameters that can be trained. For models with more nodes and more layers, the parameters will be more, and it is easier to overfit the same task. Increasing the training data also improves the overfitting, but the "train" data is fixed and there is no more data, so it is not possible here to improve the overfitting by increasing the training data. Moreover, there are also methods to solve overfitting like adding regular items, dropouts, and so on.

Therefore, to solve the over-fitting problem, we should first try to reduce the model parameters. For the neural network, we must reduce the number of nodes of the network. For this purpose we establish the model which has a structure as Fig. 14 shown. The model consists of three layers ,and every layer has 12 LSTM node and 1 fully connected nodes, a total of 4333 common network parameters, compared to the parameters of a model 33195, the model reduce the number of the original parameters by 13%.

To fit the "train" sets of training data, the model set the training set Epoch as 300, choose the ADAM optimizer, initial learning rate of 0.001, an error of change in the training process is seen in Fig. 15, the horizontal axis represents epoch , the vertical axis represents loss = MSE , and the training MSE still has a downward trend of 120 or so. Since all data is used for training, no test data is used to obtain test errors. To avoid overfitting, the Epoch upper limit is set to 300 to stop. In fact, the training error reaches MSE= 175 when Epoch gets to 170, which is already a good result. Continued training to Epoch=300 has a fairly high risk of over-fitting. The presented model uses # 2, # 8, # 111, # 147 as sample data for prediction, the result is observed in Fig. 16. The results are not much different from the previous section. , but this model predict test data and upload it to obtain NASA 900 points, ranked fifth in the reference score, which is a good result.

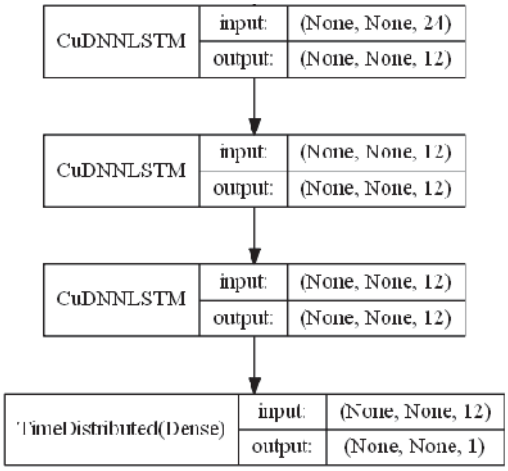


Figure 14. "Simplified" model structure and training process error changes

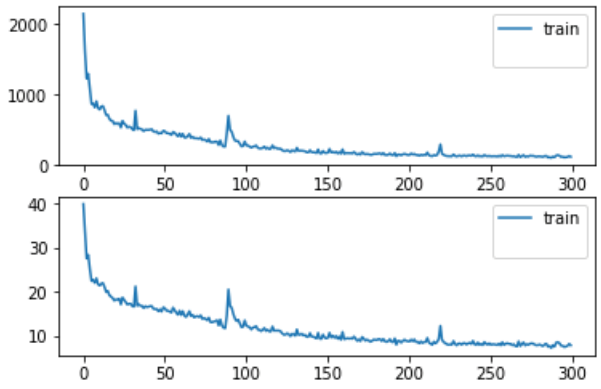
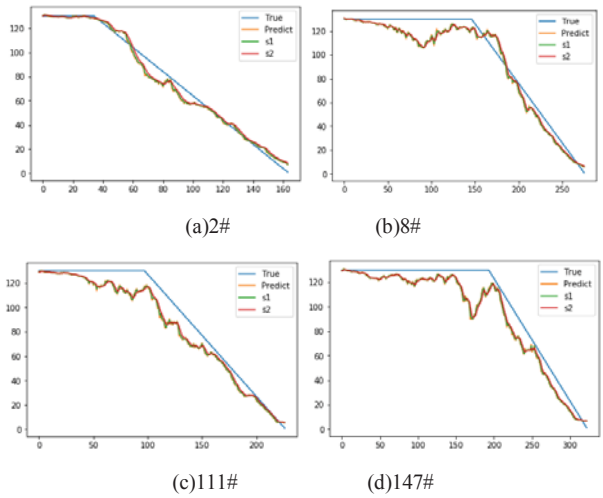


Figure 15. “Simplified” model structure and training process error changes



(a)2# (b)8#

(c)111# (d)147#

Figure 16. Simplified model prediction example

VI. CONCLUSION AND FUTURE WORK

In this paper, we use the high-dimensional sensor data of PHM08 Challenge problem scenario to study the application of LSTM deep learning method to predict the remaining life. By comparing different data input methods , it is determined that the correct method is to input the time series information all at once. In order to improve the accuracy of the prediction results, the over-fitting phenomenon is alleviated to a certain extent by reducing the complexity of the neural network. Finally, the evaluation of the test set results by the competition's scoring system shows that the LSTM network has a good effect on the remaining life prediction based on multidimensional data.

In the future, we can further study the neural network automatic construction and automatic adjustment of parameters to simplify the difficulty of building a model. In addition, we can compare the effects of different RNN deep learning methods and consider combining different methods to build a composite model with higher accuracy.

REFERENCES

[1] S. Spiewak, R. Duggirala, and K. Barnett, “Predictive Monitoring and Control of the Cold Extrusion Process,” CIRP Annals - Manufacturing Technology.U, vol. 49, no. 1, pp. 383–386, April 2000.

[2] C. Olah. “Understanding LSTM Networks”. https://colah.github.io/-posts/2015-08-Understanding-LSTMs/,2015-8-27

[3] Y. Wu, M. Yuan, S Dong, et al. “Remaining useful life estimation of engineered systems using vanilla LSTM neural networks,”. Neurocomputing, vol. 27, no. 5, pp. 167-179, June 2018.

[4] J. Chung, C. Gulcehre, K H Cho, et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling,” arXiv preprint arXiv.vol. 14, no. 12, pp. 35-55, June 2014

[5] Skymind. “ A Beginner's Guide to LSTMs and Recurrent Neural Networks”. https://skymind.ai/wiki/lstm, 2018-11-25

[6] E. Ramasso, A. Saxena. “Performance Benchmarking and Analysis of Prognostic Methods for CMAPSS Dataset,” International Journal of Prognostics and Health Management, vol. 5, no. 2, pp. 1–15, April 2014.

[7] F. Heimes, “Recurrent neural networks for remaining useful life estimation,” Prognostics and Health Management,International Conference on. IEEE, 2008: 1-6.

[8] N . Srivastava, G . Hinton, A . Krizhevsky, et al. “Dropout: a simple way to prevent neural networks from overfitting,”The Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, April 2014.

[9] X. Glorot, Y.Bengio, “Understanding the difficulty of training deep feedforward neural networks,” Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010: 249-256.

[10] M. Schrimpf, S. Merity, J. Bradbury, et al. “A Flexible Approach to Automated RNN Architecture Generation,” arXiv preprint arXiv, vol. 17, no. 12, pp. 66-73, April 2017.

[11] K. Cho, B. Merriënboer, C. Gulcehre, et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” arXiv preprint arXiv, vol. 14, no. 06, pp. 70-78, June 2014.

[12] T. Wang, J. Yu, D. Siegel, et al. “A similarity-based prognostics approach for remaining useful life estimation of engineered systems,” Prognostics and Health Management, International Conference on. IEEE, 2008: 1-6.

[13] T. Wang, Trajectory similarity based prediction for remaining useful life estimation, CA: University of Cincinnati, 2010.