The Journal of Supercomputing (2019) 75:7592–7605
https://doi.org/10.1007/s11227-019-02967-7

Host load prediction in cloud computing using Long Short-Term Memory Encoder–Decoder

Hoang Minh Nguyen[1] · Gaurav Kalra[1] · Daeyoung Kim[1]

Published online: 19 August 2019

Abstract
Cloud computing has been developed as a means to allocate resources efficiently while maintaining service-level agreements by providing on-demand resource allocation. As reactive strategies cause delays in the allocation of resources, proactive approaches that use predictions are necessary. However, due to high variance of cloud host load compared to that of grid computing, providing accurate predictions is still a challenge. Thus, in this paper we have proposed a prediction method based on Long Short-Term Memory Encoder–Decoder (LSTM-ED) to predict both mean load over consecutive intervals and actual load multi-step ahead. Our LSTM-ED-based approach improves the memory capability of LSTM, which is used in the recent previous work, by building an internal representation of time series data. In order to evaluate our approach, we have conducted experiments using a 1-month trace of a Google data centre with more than twelve thousand machines. Our experimental results show that while multi-layer LSTM causes overfitting and decrease in accuracy compared to single-layer LSTM, which was used in the previous work, our LSTM-ED-based approach successfully achieves higher accuracy than other previous models, including the recent LSTM one.

Keywords Host load prediction · Cloud computing · Long Short-Term Memory Encoder–Decoder

✉ Hoang Minh Nguyen
minhhoang@kaist.ac.kr

Gaurav Kalra
gvkalra@kaist.ac.kr

Daeyoung Kim
kimd@kaist.ac.kr

[1] School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea

1 Introduction

As computers and data centres are getting more sophisticated and demanding more power than ever, efficient resource allocation while maintaining service-level agreements (SLAs) has become increasingly important. Thus, the cloud computing model has been developed, which enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [11].

For services in the cloud, resources can be autoscaled in either reactive or proactive methods. Reactive methods monitor resources usage and make resources allocation and deallocation decisions on-the-fly based on certain load thresholds. Proactive methods, on the other hand, are able to make decisions before overloading or underloading situations happen by predicting future resources usage. As allocation or deallocation of resources can take up to minutes [10], proactive methods, rather than reactive ones, are necessary to deal with the delay.

Host load prediction is essential to enable these proactive methods; as a result, there have been many prior host load prediction works, most of which focused on grid/HPC systems [5, 9, 17]. However, by studying workload traces of a production data centre at Google [6], Google host load was found to exhibit finer resource allocation with respect to CPU and memory than those of grid/HPC systems [3]. Google jobs are much shorter and always submitted with much higher frequency than Grid jobs. Thus, Google host load exhibits higher variance and noise.

Due to drastic fluctuations at small timescales, the Google host load traces [6] have attracted a series of works with different prediction approaches. The traditional methods of autoregressive (AR) [17] and Artificial Neural Networks (ANN) [5] were commonly applied for grid datasets; however, they show poor performance in mean load prediction for the Google dataset. The Bayesian approach [4] could only predict mean host load values, and the Phase Space Reconstruction and Group Method of Data Handling based on Evolutionary Algorithm [18] is limited in utilizing long-term historical values. To improve the multi-step-ahead host load prediction capability, the Autoencoder and Echo State Networks (ESN) method [19] used an Autoencoder to extract input features and a nonlinear ESN reservoir to obtain predictions. Still, this method has limited generalization ability due to manually chosen parameters in ESN and random reservoir initialization, and highly depends on the features extracted from the Autoencoder. Thus, the current state-of-the-art method of Long Short-Term Memory (LSTM) [13] further improved prediction accuracy through parameters auto-tuning and powerful capability of long-term dependencies learning.

In this work, to improve upon the current state-of-the-art LSTM method, we have utilized a method called Long Short-Term Memory Encoder–Decoder (LSTM-ED) to perform multi-step-ahead host load prediction. Similar to LSTM, LSTM-ED can automatically learn long-term dependencies without any extra features extracted step; however, different from LSTM, LSTM-ED can improve prediction accuracy by building a representation of the time series data. Firstly, we have shown that multi-layer LSTM causes overfitting and has poor performance for both mean load and actual load prediction tasks. Secondly, we show that building a representation from the input data by optimizing LSTM-ED can provide accurate future mean load and actual load predictions for the Google load traces dataset. In fact, experimental results on the Google dataset show that our method achieves higher accuracy than current state-of-the-art methods.

The remainder of the paper is organized as follows. Section 2 introduces the related work to ours. Section 3 provides explanations about our method, including architecture overview, Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Long Short-Term Memory Encoder–Decoder (LSTM-ED). Section 4 presents our experimental results, including the Google load traces used in evaluation, accuracy metrics, results of single- and multi-layer LSTM, and mean and actual load prediction results of our method. Finally, Sect. 5 provides concluding remarks for this paper. In this paper, 'autogressive' and 'autoregression' are used interchangeably.

2 Related work

Host load prediction in grid and cloud systems has gathered a lot of attention from researchers due to its benefits in improving resource allocation and utilization while satisfying service-level agreement (SLA).

Prior works mostly focused on grid and HPC systems, which were improved from the works on traditional distributed systems. Li et al. [9] made a survey of load balancing in grid computing, where they classified load balancing schemes into three categories, including resource-aware repartition, divisible load theory, and prediction-based schemes. Wu et al. [17] developed a hybrid method which combined autoregressive (AR) model with confidence interval estimations to predict and Kalman filter to eliminate noise. Their model greatly improved prediction accuracy compared to conventional AR model and was able to perform multi-step-ahead prediction. Still, AR is a classical method that has poor performance in long-term and mean load prediction. Duy et al. [5] greatly reduced prediction errors by applying a Feedforward Artificial Neural Network (ANN); however, their model was not able to predict long-term host load due to a limited number of nodes in the network.

However, the Google host load [6] exhibits finer resource allocation, has higher frequency of jobs, and shows higher variance and noise than grid host load [3]. These characteristics make it difficult for traditional methods to make accurate predictions due to their lack of feature extraction ability. As a result, this challenge has motivated multiple works on providing accurate multi-step-ahead predictions for the Google traces. The first notable work was from Di et al. [4], where they extracted 9 features from the traces and selected appropriate features to combine in their Bayes model. Their model was able to provide multi-step-ahead predictions and achieved higher accuracy than traditional models like autoregression (AR); however, the predictions were only for mean host load values with exponentially growing segments' lengths.

Yang et al. [18] combined Phase Space Reconstruction (PSR) and Group Method of Data Handling based on Evolutionary Algorithm (EA-GMDH). PSR was used to reconstruct the time series and EA-GMDH uses a Feedforward Neural Network optimized by Evolutionary Algorithm to make predictions. However, the method was not able to make multi-step-ahead predictions due to its limited number of nodes. In [19], an Autoencoder was used to extract features from the time series and Echo State Networks (ESN) were used to perform multi-step-ahead host load prediction. Even though this method outperformed previous approaches, it still has several shortcomings of limited generalization ability due to manually chosen leaking rate, high dependency on features extracted by the Autoencoder, and possible performance degradation due to random initialization of large reservoir.

In order to learn long-term dependencies to make accurate multi-step-ahead host load predictions, Song et al. [13] utilized the Long Short-Term Memory (LSTM) Recurrent Neural Network in their method. LSTM has three major advantages over previous approaches, including ability to learn long-term dependencies, automatic features extraction, and powerful nonlinear generalization capability.

In this work, we have first presented the limits of LSTM by showing that multi-layer LSTM causes overfitting in host load prediction. As a result, to further improve upon the memory capability of single-layer LSTM, we have used the Long Short-Term Memory Encoder–Decoder (LSTM-ED) for host load prediction. LSTM-ED further improves LSTM's ability to learn long-term dependencies by building an internal representation of the host load data, while still retains LSTM's advantages of automatic features extraction and powerful nonlinear generalization ability. Our experimental results clearly demonstrate that our method outperforms all previous methods in cloud host load prediction.

3 Our method

3.1 Architecture overview

Figure 1 shows the architecture overview of our method. At the centre of our method, we use a model called Long Short-Term Memory Encoder–Decoder (LSTM-ED) for

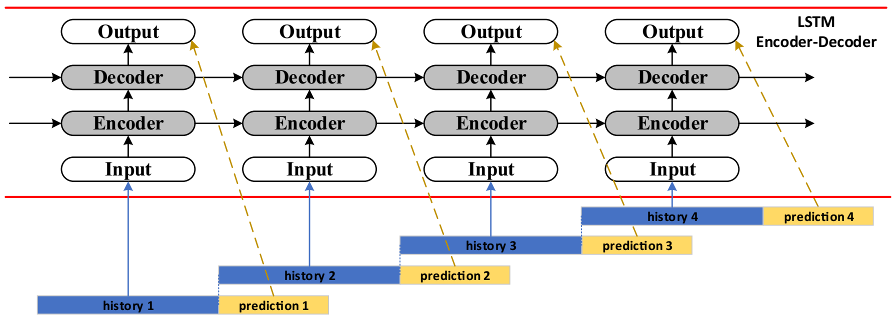


Fig. 1 Architecture overview

host load prediction. The LSTM-ED consists of two main components, including an Encoder which builds a representation that encapsulates information from the time series data, and a Decoder which 'decodes' the built representation into outputs.

The host load time series is divided into consecutive 'history' sequences of fixed size; each of this 'history' sequence is accompanied by a 'prediction' sequence of fixed size. The 'history' and 'prediction' sequences are used as inputs and supervised outputs/labels for the LSTM-ED, respectively. Depending on the performed tasks, these 'prediction' sequences can be either real host load values or mean load values over future time intervals.

In addition, the Google load trace provides measurements taken at 1-s intervals, which is too small compared to the usual CPU load fluctuations in the trace. Thus, in this work the smallest interval we have used to take samples from the trace is 5 min, which is the same as previous works that used this case for comparison purpose.

3.2 Recurrent Neural Network

Recurrent Neural Network is a type of network with loops in them to allow information to persist. This type of network can be used for our host load prediction task by predicting the next value based on historical load values. Figure 2 shows the architecture of a Recurrent Neural Network (RNN) with one hidden layer together with its unrolled form, in which at time step t in the network x_t is the input, s_t is the hidden state, and o_t is the output.

In our host load prediction task, x_t can be the historical load value (possibly after normalization). Then the hidden state s_t of RNN can be calculated based on the previous hidden state and the output at the current time step:

$$s_t = \sigma(Ux_t + Ws_{t-1}) \tag{1}$$

where σ is usually a nonlinear function like *tanh* or *ReLU*. To calculate the first hidden state, s_{-1} is typically initialized to zeros.

The output state o_t can be calculated based on the hidden state s_t as follows:

$$o_t = Vs_t \tag{2}$$

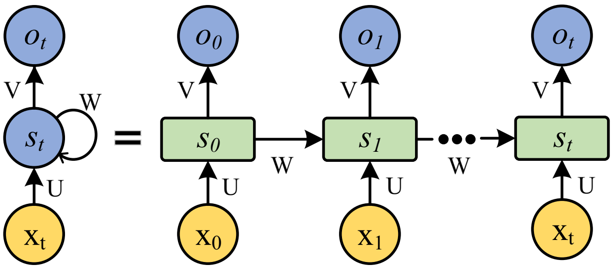


Fig. 2 An unrolled Recurrent Neural Network

Different from a traditional deep neural network, RNN uses the same set of parameters (U, V, W) across all steps, which greatly reduces the number of parameters the network needs to learn.

The most well-known training algorithm of RNN is backpropagation through time [15], which consists of repeated applications of the chain rule. However, as discovered by Hochreiter [7] and Bengio et al. [1], RNN suffers from two major problems of exploding and vanishing gradients.

The exploding gradients issue arise when the weights are assigned values which are too high; this issue can easily be solved by truncating or squashing the gradients. The vanishing problem arises when the gradients' values become too small and the RNN model learns very slowly or even stops learning altogether; this issue is much harder to solve than the exploding gradients one. It was not until 1997 that this issue was solved through the concept of Long Short-Term Memory (LSTM) introduced by Hochreiter et al. [8].

3.3 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a special kind of RNN, which can resolve the vanishing gradients issue and is capable of learning long-term dependencies. Introduced by Hochreiter et al. [8] in 1997, there have been many works which apply LSTM. These include the recent previous work by Song et al. [13], where they optimized the parameters and showed that their LSTM model outperformed other previous models in the Google load trace.

Figure 3 illustrates the architecture overview of a LSTM cell, which consists of the following components:

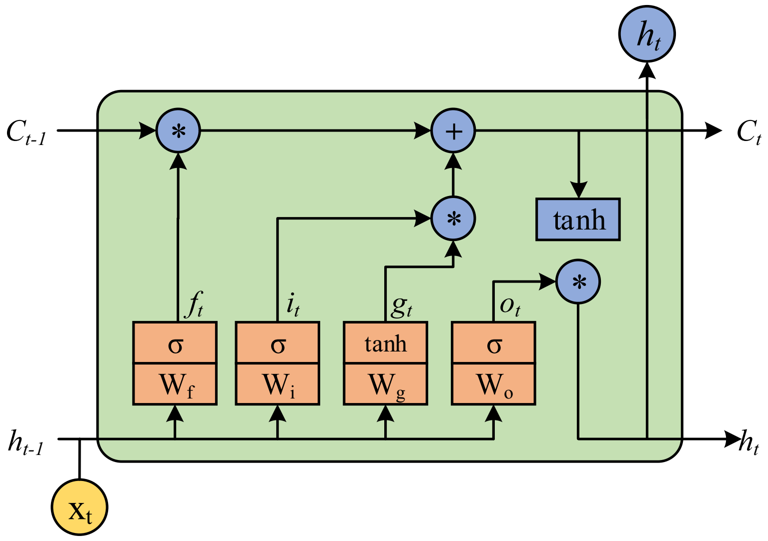


Fig. 3 LSTM cell

- x_t: external input at time step t.
- h_{t-1}, h_t: hidden state at times $(t-1)$ or t. This is also used as output or input for the next layer of LSTM cells (in multi-layer LSTM).
- C_{t-1}, C_t: the 'cell state' or 'memory' at time step $t-1$ or t.
- f_t: the result of the forget gate, which controls whether to forget (for values close to zero) or remember (for values close to one) the memory C_{t-1}.
- i_t: the result of the input gate, which determines the degree of importance of the (transformed) new external input.
- g_t: the result of the cell gate, which performs a nonlinear transformation of the new external input x_t.
- o_t: the result of the output gate, which controls the amount of the new cell state C_t that goes to the output and the hidden state.

Every time the LSTM takes an input x_t, the four gates f_t, i_t, g_t, and o_t are updated as follows:

$$\begin{aligned} f_t &= \sigma(W_f * [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i * [h_{t-1}, x_t] + b_i) \\ g_t &= tanh(W_g * [h_{t-1}, x_t] + b_g) \\ o_t &= \sigma(W_o * [h_{t-1}, x_t] + b_0) \end{aligned} \tag{3}$$

where we use sigmoid function for σ in this work.

The cell state at the current time step can be updated using the results from the cell gate and the cell state at the last time step as follows:

$$C_t = f_t * C_{t-1} + i_t * g_t \tag{4}$$

Finally, the hidden state (or output) can be updated using the results from the output gate and the cell state at the current time step as follows:

$$h_t = o_t * tanh(C_t) \tag{5}$$

3.4 Long Short-Term Memory Encoder–Decoder

In order to further improve upon the previous work's LSTM model that shows long-term dependencies learning capability, in this work we have used a model called Long Short-Term Memory Encoder–Decoder (LSTM-ED). The LSTM-ED model consists of two LSTM RNN that act as an Encoder and a Decoder pair, as illustrated in Fig. 4.

The LSTM Encoder–Decoder was developed for natural language processing tasks and demonstrated state-of-the-art performance [2]. At the heart of this model is a compressed representation that input sequences are read to and output sequences are read from. Different from the LSTM model, LSTM-ED reads the entire input sequence into an internal representation which is a fixed-length vector. This internal representation acts as the last hidden state of the Encoder as well as the first hidden state of the Decoder. This internal 'memory' allows information to be remembered across long input sequences. For this reason, the model

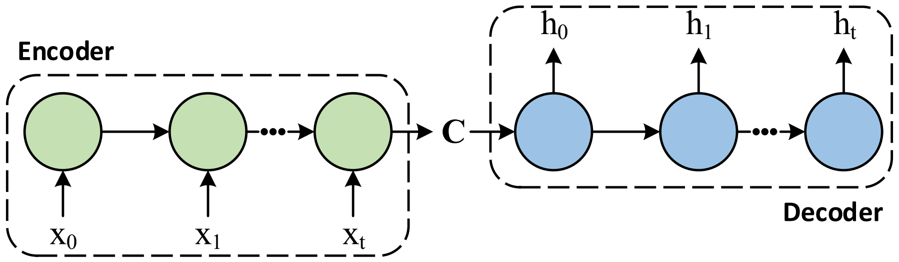


Fig. 4 Long Short-Term Memory Encoder–Decoder

is also referred to as sequence embedding, and it was shown to be effective for very long input sequences [14].

One potential drawback of using LSTM, which includes our LSTM-ED model, is that it requires considerable GPU resources and sufficient time for model training. This can be different from some older methods where the use of CPU may be sufficient for model training (e.g. Echo State Network); however, the prediction accuracy of LSTM outweighs those methods that only need CPU for training [13].

3.5 Parameters of our method

To train our network, we have used the backpropagation through time (BPTT) algorithm, which consists of repeated applications of the chain rule [15]. Similar to the previous Google host load prediction work using LSTM [13], we 'clip' the gradient before parameters update when it becomes to large to prevent exploding gradients [12]. We have also used similar input layer size, hidden layer size, batch size, number of epochs, and learning rate, which is also annealed by 0.1 every 30 epochs.

In addition, due to the long length of our prediction method, we have used 'truncated backpropagation through time' [16] to reduce the cost of a single parameter update, similar to the previous LSTM work [13]. The truncated length is kept at 39 for the mean load prediction task; however, different from the previous LSTM work, this length is reduced to 26 for the real values prediction task due to long-term dependencies learning ability of LSTM-ED method. The parameters' values of our LSTM-ED method are summarized in Table 1.

Table 1 Parameters of our LSTM-ED method

Parameter	Value
Input layer size	24/64
Hidden layer size	128
Batch size	128
Global gradient clipping norm	5
Truncated length	26/39
Epoch	90
Learning rate	0.05

4 Experimental results

In order to evaluate our method, we have predicted both the actual load value and mean load value of the Google cluster workload traces [6]. Before training of our method and all other benchmark methods, the input data are standardized by removing the mean and scaling to unit variance to help with the convergence of gradient descent and performance of the methods.

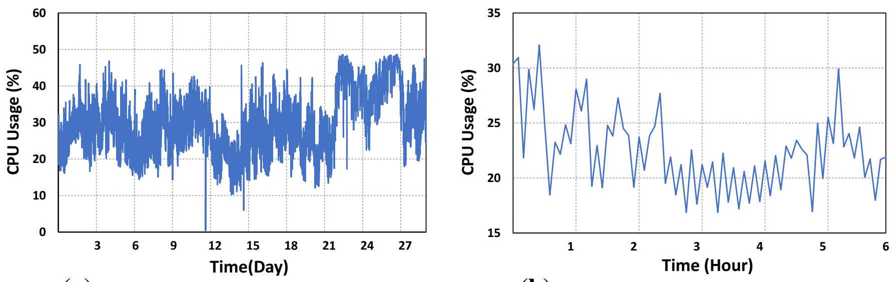
4.1 Google load traces and benchmark models

The Google cluster workload traces [6] contain over 25 million tasks across over 12,500 hosts during 29 days in May 2011. The dataset was split into three sets, similar to previous methods [13, 18, 19]. These include a training set (from the beginning to the 20th day) to calculate LSTM-ED's weights, a validation set (from the 21st to the 26th day) to choose hyperparameters and prevent overfitting, and a testing set (from the 27th day to the end) to evaluate our model. After the hyperparameters were confirmed, the training and validation sets were combined together to train a final model, and the final model was evaluated on the testing set.

In order to evaluate the accuracy of our method, we have compared it against previous notable methods including autoregressive (AR) [17], Artificial Neural Networks (ANN) [5], Bayesian [4], Phase Space Reconstruction and Group Method of Data Handling based on an Evolutionary Algorithm (PSR+EA-GMDH) [18], Autoencoder and Echo State Networks (AE-ESN) [19], and Long Short-Term Memory Recurrent Neural Network (LSTM) [13]. Similar to the other methods, we have only provide predictions for CPU load using only CPU usage values. As an example, the host load of one machine is shown in Fig. 5.

4.2 Accuracy metrics

In order to compare our method with previous works, we have performed both mean load and actual load prediction. For mean load prediction, we have used a metric called exponentially segmented pattern (ESP) [4] to characterize host load fluctuation over consecutive time intervals whose lengths increase exponentially.



(a) Host load of the whole 29 days (b) Host load of the first 6 hours

Fig. 5 Host load of a single machine (id 563849022) in Google load traces

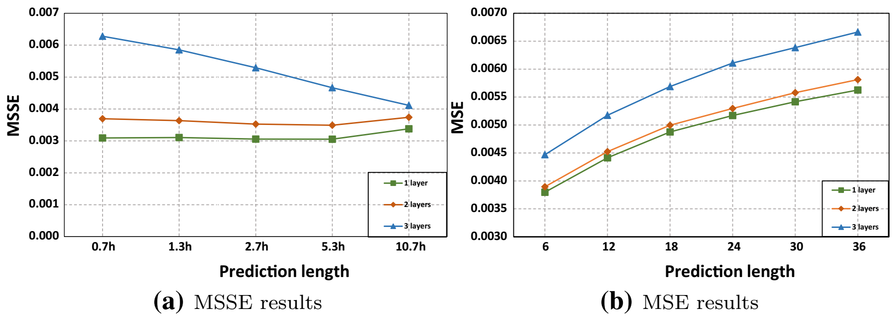


Fig. 6 Accuracy results of 1, 2, and 3 layers of LSTM for 1000 hosts randomly selected from the Google dataset

To quantify the accuracy of mean load prediction over ESP, the mean segment squared error (MSSE) is defined as follows:

$$\mathrm{MSSE}(s) = \frac{1}{s} \sum_{i=1}^{n} s_i (l_i - L_i)^2 \tag{6}$$

where l_i is the predicted mean value, L_i is the true mean value, $s_1 = b$, $s_i = b * 2^{i-2}$, $s = \sum_{i=1}^{n} s_i$, and b is called the baseline segment, which is set to 5 min throughout this work. In practice, this load pattern prediction is converted from single load interval prediction [4].

Still, the mean load over a segmented pattern may not be able to fully capture host load variation, which is important in evaluating the capability of a prediction method. Thus, similar to [13], we have also used mean squared error (MSE) to evaluate the accuracy of our method:

$$\mathrm{MSE} = \frac{1}{N} \sum_{i=1}^{N} (o_i - y_i)^2 \tag{7}$$

where o_i is the prediction value, y_i is the true load value, N is the prediction length, and the interval length between two consecutive host load values is set to 5 min for the Google dataset.

4.3 Single- and multi-layer Long Short-Term Memory

As the previous work uses LSTM with only a single layer [13], we have performed experiments to compare between single-layer and multi-layer LSTM. Due to the heavy computation required from deep LSTM with multiple layers, we have randomly selected 1000 hosts from the Google dataset to run our experiments with 1, 2, and 3 layers of LSTM.

Figure 6 shows the MSSE and MSE results of 1, 2, and 3 layers of LSTM. It is clear that deep LSTM with more than a single layer generates overfitting; higher number of LSTM layers causes bigger decrease in prediction accuracy. It is also worth noting that while 2-layer LSTM has relatively close results to those of single-layer LSTM, 3-layer LSTM causes a significant decrease in prediction accuracy compared to that of the other two.

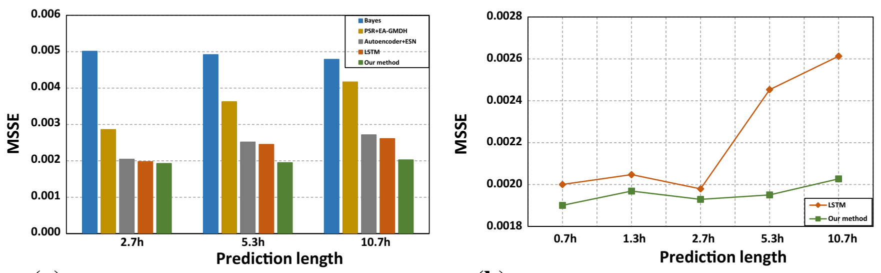
In addition, in Fig. 6a, the average MSSE results of 3-layer LSTM actually improve for longer prediction lengths due to the MSSE calculation placing higher weights on longer prediction length. As there are less correlations between historical data and long-term prediction values and mean load values are relatively easier to predict than actual load values, the overfitting issue of 3-layer LSTM model becomes less significant, and its long-term MSSE results ‘average out’. This ‘averaging effect’ can be seen when compared to the MSE results in Fig. 6b, where all prediction lengths are treated equally, and the differences among 1-, 2-, and 3-layer LSTM are similar for varied prediction lengths. Overall, the results in Fig. 6 clearly demonstrate that deep LSTM model with multiple layers does not perform well for the Google load traces.

4.4 Mean load prediction

As discussed in [4], there were traditional methods of AR [17] and ANN [5], but they show poor accuracy in mean load prediction. As a result, we have compared our method with four other methods, including Bayesian [4], PSR+EA-GMDH [18], Autoencoder+ESN [19], and LSTM [13] that show good performance in mean load prediction.

Figure 7 shows the comparison of MSSE results among different methods. As can be seen in Fig. 7a, Autoencoder+ESN, LSTM, and our method significantly outperform Bayes and PSR+EA-GMDH. Our method also achieves the best accuracy among all five methods for all prediction lengths.

A detailed comparison between our method and the previous state-of-the-art LSTM model [13] is shown in Fig. 7b. While the differences in accuracy between our method and LSTM are not too significant for prediction lengths 0.7, 1.3, and 2.7 h, the differences for very long-term prediction lengths of 5.3 and 10.7 h are quite significant. As a result, the MSSE–length curve of our method is much



(a) Comparison between our method and four others for consecutive intervals

(b) Comparison between our method and LSTM

Fig. 7 MSSE results comparison among different methods with baseline segment of 5 min

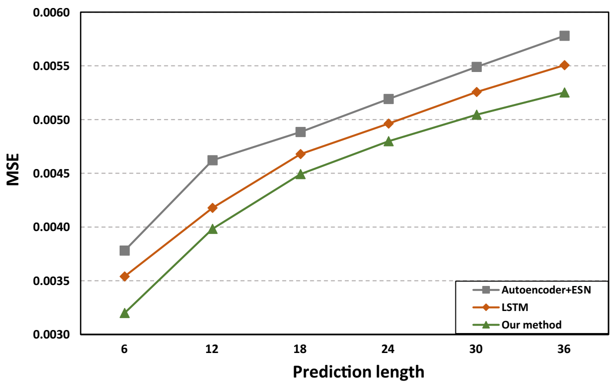


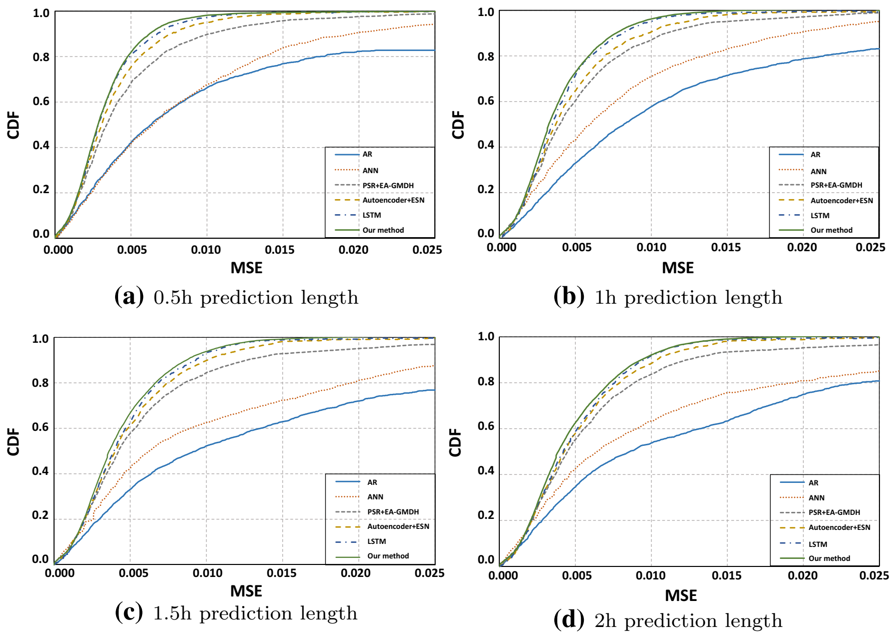
Fig. 8 MSE results comparison among three methods with interval length of 5 min

'smoother' than that of LSTM, which clearly demonstrates the better nonlinear generalization ability of LSTM-ED to deal with the high variance and noise of host load.

4.5 Actual load prediction

To fully evaluate the capability of our method in dealing with the variation of host load, we have compared the MSE results of our method with the current two state-of-the-art methods of Autoencoder+ESN [19] and LSTM [13] and provided the Cumulative Distribution Function (CDF) of MSE results of our method and five other methods mentioned throughout the paper, including AR [17], ANN [5], PSR+EA-GMDH [18], Autoencoder+ESN [19], and LSTM [13]. In Fig. 9, each point in the CDF lines shows the ratio of results (vertical axis) that are less than or equal to the corresponding MSE value (horizontal axis). Thus, the vertical axis of each CDF line is bounded between 0 and 1, and the closer a CDF line is to the vertical axis, the higher accuracy (or lower MSE) the corresponding model has.

Figure 8 clearly shows that our method consistently achieves higher accuracy than the other two methods for all prediction lengths. Similar to LSTM, our method, which uses LSTM-ED, can extract more useful features than the Autoencoder and the reservoir of ESN and does not cause cumulative errors due to vector instead of single output. However, different from LSTM, our method generates a representation to encapsulate features from the time series, which allows it to have a better nonlinear generalization ability than LSTM. This is also demonstrated in Fig. 9, where the CDF lines of our method show a better MSE distribution than all other five methods for all prediction lengths.



(a) 0.5h prediction length

(b) 1h prediction length

(c) 1.5h prediction length

(d) 2h prediction length

Fig. 9 Cumulative Distribution Function (CDF) of MSE results among six different methods

5 Conclusion

We have proposed a method based on Long Short-Term Memory Encoder–Decoder (LSTM-ED) for host load prediction in the cloud. In addition, we have performed evaluations using the Google load traces in terms of mean load prediction over consecutive intervals and actual load prediction over single intervals. The experimental results show that while multi-layer LSTM causes overfitting and performs worse than single-layer LSTM, our method is able to achieve better accuracy than all previous methods for all prediction lengths in both mean load and actual load prediction. In case of mean squared error (MSE), our model is able to attain MSE results of 0.0032, 0.0039, 0.0044, 0.0047, 0.0050, and 0.0052 for prediction lengths of 6, 12, 18, 24, 30, and 36, respectively. In case of mean segment squared error (MSSE), our model is able to attain MSSE results of 0.00190, 0.00196, 0.00192, 0.00195, and 0.00202 for prediction lengths of 4, 5, 6, 7, and 8, respectively.

For our future works, we will conduct more research to further improve the memory capability of LSTM-ED and improve the host load prediction accuracy. As our model is able to achieve high accuracy for different prediction lengths, we plan to deploy our model and conduct experiments on cloud settings, including both simulations and real-world environments.

Acknowledgements This research was supported by International Research and Development Program of the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning of Korea (2016K1A3A7A03952054), and Smart City R&D project of the Korea Agency for

Infrastructure Technology Advancement (KAIA) Grant funded by the Ministry of Land, Infrastructure and Transport (MOLIT), Ministry of Science and ICT (MSIT) (Grant 18NSPS-B149386-01).

References

1. Bengio Y, Simard P, Frasconi P et al (1994) Learning long-term dependencies with gradient descent is difficult. IEEE Trans Neural Netw 5(2):157–166
2. Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder–decoder for statistical machine translation. arXiv:14061078
3. Di S, Kondo D, Cirne W (2012) Characterization and comparison of cloud versus grid workloads. In: IEEE International Conference on Cluster Computing (CLUSTER), 2012. IEEE, pp 230–238
4. Di S, Kondo D, Cirne W (2012) Host load prediction in a Google compute cloud with a Bayesian model. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press, p 21
5. Duy TVT, Sato Y, Inoguchi Y (2011) Improving accuracy of host load predictions on computational grids by artificial neural networks. Int J Parallel Emerg Distrib Syst 26(4):275–290
6. Google cluster workload traces (2011). https://github.com/google/cluster-data
7. Hochreiter S (1991) Untersuchungen zu dynamischen neuronalen netzen. Diploma, Technische Universität München, 91(1)
8. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
9. Li Y, Lan Z (2004) A survey of load balancing in grid computing. In: International Conference on Computational and Information Science. Springer, pp 280–285
10. Lorido-Botrán T, Miguel-Alonso J, Lozano JA (2012) Auto-scaling techniques for elastic applications in cloud environments. Department of Computer Architecture and Technology, University of Basque Country, Tech Rep EHU-KAT-IK-09 12:2012
11. Mell P, Grance T et al (2009) The NIST definition of cloud computing. Natl Inst Stand Technol 53(6):50
12. Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks. In: International Conference on Machine Learning, pp 1310–1318
13. Song B, Yu Y, Zhou Y, Wang Z, Du S (2018) Host load prediction with long short-term memory in cloud computing. J Supercomput 74(12):6554–6568
14. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp 3104–3112
15. Werbos PJ et al (1990) Backpropagation through time: what it does and how to do it. Proc IEEE 78(10):1550–1560
16. Williams RJ, Peng J (1990) An efficient gradient-based algorithm for on-line training of recurrent network trajectories. Neural Comput 2(4):490–501
17. Wu Y, Yuan Y, Yang G, Zheng W (2007) Load prediction using hybrid model for computational grid. In: 8th IEEE/ACM International Conference on Grid Computing, 2007. IEEE, pp 235–242
18. Yang Q, Peng C, Zhao H, Yu Y, Zhou Y, Wang Z, Du S (2014) A new method based on PSR and EA-GMDH for host load prediction in cloud computing system. J Supercomput 68(3):1402–1417
19. Yang Q, Zhou Y, Yu Y, Yuan J, Xing X, Du S (2015) Multi-step-ahead host load prediction using autoencoder and echo state networks in cloud computing. J Supercomput 71(8):3037–3053