Log-Based Anomaly Detection with Multi-Head Scaled Dot-Product Attention Mechanism

Qingfeng Du(✉), Liang Zhao, Jincheng Xu, Yongqi Han, and Shuangli Zhang

School of Software Engineering, Tongji University, Shanghai, China
{du_cloud,1931525,xujincheng,2011438,1931551}@tongji.edu.cn

Abstract. Anomaly detection is one of the key technologies to ensure the performance and reliability of software systems. Because of the rich information provided by logs, log-based anomaly detection approaches have attracted great interest nowadays. However, it's time-consuming to check the large amount of logs manually due to the ever-increasing scale and complexity of the system. In this work, we propose a log-based automated anomaly detection approach called *LogAttention*, which embeds log patterns into semantic vectors and subsequently uses a self-attention based neural network to detect anomalies in the log pattern sequences. LogAttention has the ability to capture contextual and semantic information in the log patterns and to attend far more long-range dependencies in the log pattern sequence. We evaluate LogAttention on two publicly available log datasets, and the experimental results demonstrate that our proposed approach can achieve better results compared to the existing baselines.

Keywords: Anomaly detection · Log analysis · Log data · Attention · Deep learning

1 Introduction

Anomaly detection is an essential task to ensure the reliability and the performance of software systems, which aims to detect anomalies in time to avoid further failures and losses. As the system gets increasingly complex and error-prone, there is an urgent need for the effective approaches of anomaly detection.

Anomaly detection approaches can be divided into several categories from different views. Log-based anomaly detection approaches have attracted great interest in recent years because of the rich information in logs. System logs can accurately describe the system status and events. Therefore, they can provide the necessary support for system diagnosing. Logs are usually viewed as plain texts with two essential components: constant parts and variable parts [7]. In this paper, we refer to the constant parts as *log patterns*.

With the ever-increasing scale and complexity of software systems, it's not easy to manually detect anomalies from logs [16]. Up to now, many automatic

C. Strauss et al. (Eds.): DEXA 2021, LNCS 12923, pp. 335–347, 2021.
https://doi.org/10.1007/978-3-030-86472-9_31

approaches [1,10] have been proposed to ease the pressure of manual analysis. Most of them need to extract log pattern sequences from raw logs. According to how log pattern sequences are vectorized, these approaches can be further classified into two categories, including log pattern counter based approaches and deep learning based approaches [12]. Log pattern counter based approaches suffer from the dependency on the parsing accuracy. In recent years, deep learning based approaches using Long Short-Term Memory (LSTM) have enjoyed tremendous success [5,12,16]. Instead of using the log pattern count vector, they extract sequential features hidden in the log sequence with recurrent structures. Nonetheless, it is difficult for LSTM to accurately learn the dependency when the path between two interconnected log patterns becomes too long. These problems pose major obstacles for the development of automatic approaches.

In order to overcome these obstacles, we propose a log-based automatic anomaly detection approach called *LogAttention*, which has the ability to learn the semantic information and long-range dependencies effectively in real-world logs. Firstly, LogAttention leverages Drain [6] to extract log patterns from raw logs. Then LogAttention generates semantic vectors for log patterns by aggregating multiple word vectors based on TF-IDF scores. The semantic vectors can reduce the dependency on parsing accuracy. The core part of LogAttention is *multi-head scaled dot-product attention* [14], which takes the sequence of log pattern vectors as input and subsequently detects the possible anomalies. Multi-head scaled dot-product attention is able to learn long-range dependencies in the log pattern sequence and each head can be trained in parallel.

In this work, we evaluate LogAttention on two publicly available datasets which collect logs from Hadoop system and Blue Gene/L supercomputer respectively. The overall experimental results show that our proposed approach can effectively detect anomalies in real systems compared with existing baselines. In addition, we use different window lengths to slice the logs of BGL and evaluate LogAttention's learning ability of long-range dependencies. The results show that LogAttention performs even better when detecting anomalies in the longer log sequences. Finally, we evaluate the effectiveness of the proposed vectorization method.

Our main contributions are listed below:

1. We propose a new vectorization method for log patterns, the multiple word vectors of which are aggregated based on TF-IDF to distinguish the different levels of word importance for anomaly detection.
2. We propose LogAttention, an end-to-end approach using multi-head scaled dot-product attention to detect anomalies effectively.
3. We perform extensive experiments to evaluate LogAttention on two well-established datasets.

2 Related Works

Detecting anomalies in logs can be divided into four steps: log collection, log parsing, feature extraction, and anomaly detection [7]. In this work, we concentrate on log parsing and anomaly detection.

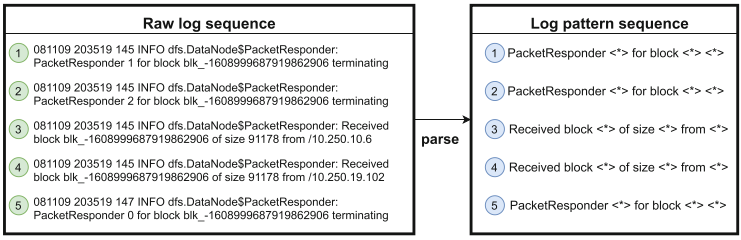

Fig. 1. An example of the raw log sequence and the extracted log patterns.

2.1 Log Parsing

Each log contains a constant part and a variable part [16]. Log parsing extracts log patterns (or the constant parts) from the raw logs. Figure 1 shows the example of the raw log sequence consisting of 5 consecutive logs and the extracted log patterns. In [17], 13 approaches on 16 datasets from various systems are comprehensively evaluated. The online approach Drain [6] shows the best performance and stability among all the considered approaches. In order to speed up the parsing process, fixed depth tree is adopted to encode parsing rules in Drain.

2.2 Log-Based Anomaly Detection

We classify existing log-based anomaly detection approaches into three groups:(1) Supervised machine learning approaches; (2) Unsupervised machine learning approaches; (3) Zero-positive machine learning approaches.

Supervised Machine Learning Approaches
A lot of supervised machine learning approaches have been applied to detect anomalies in literature. SVM (Support Vector Machines) is adopted to detect anomalies in [10]. In [16], an empirical study has been conducted to show the instability of logs in practice, and subsequently LogRobust is proposed to detect anomalies in unstable logs.

Unsupervised Machine Learning Approaches
In [15], PCA (Principle Component Analysis) is used to detect anomalies. In [11], Invariant Mining is used to detect anomalies. LogLens[3] can automatically detect anomalies with no (or minimal) knowledge of the target system.

Zero-Positive Machine Learning Approaches
Zero-positive refers to the approaches which only require normal data for anomaly detection [4]. DeepLog [5] uses LSTM to learn patterns from normal logs. It detects anomalies when logs deviates from normal distribution. LogAnomaly [12] detects sequential and quantitative anomalies at the same time. In [1], LSTM is used to learn temporal correlations and detect performance anomalies in logs.

Limitation of Existing Approaches
For those approaches based on the log pattern counter such as [10,11,15], they transform a log pattern sequence into a count vector in which each dimension represents the occurrence of a certain log pattern. The vectorization method poses the following threats. On the one hand, the count vector won't change even if the log sequence is inversed. In other words, the vector is not sensitive to the order of log sequence, while some anomalies are just manifested in the order which are called sequential anomalies. On the other hand, they are very sensitive to the results of log parsing, because the wrong log pattern will be assigned another index, which may lead to unexpected behaviors.

For those approaches based on deep learning such as [5,12,16], they can overcome the drawbacks mentioned above because they do not rely on count vectors. Besides, semantic vectorization of log patterns proposed in [16] and [12] can solve the problem of unseen log patterns. However, these approaches still suffer from the following threat: it's difficult for LSTM to accurately learn the dependencies in the longer log sequences.

3 LogAttention Design

3.1 Overview

To overcome the drawbacks of existing approaches, we propose LogAttention to effectively detect anomalies in diverse systems. The design of LogAttention has been shown in Fig. 2. In the log parsing step, we use Drain [6] to extract log patterns. After that, log patterns are transformed into semantic vectors to reduce the influence of log parsing errors. Semantic vector sequences will be fed into an multi-head scaled dot-product attention based neural network to detect whether the log sequence is anomalous or not. Each log pattern vector can attend to all the other log pattern vectors in the input sequence. In this way, anomaly features of log sequence will be automatically learned from contextual information.

3.2 Log Parsing

In log parsing, constant parts are separated from raw logs and variable parts are masked with $<*>$ [7]. For example, raw log "Received block blk_-1608999687919 of size 91178 from /10.250.10.6" can be interpreted as "Received block $<*>$ of size $<*>$ from $<*>$", which is a valid log pattern. In this work, we directly introduce Drain [6] to extract these log patterns for its performance and stability. Similar to existing log parsing approaches, Drain can not achieve 100% accuracy on any datasets, let alone incomplete logs caused by accidents. Parsing errors may result in missing necessary words in constant parts or adding redundant words into constant parts. For example, "Received block blk_-1608999687919 of size 91178 from /10.250.10.6" may be transformed into "Received block $<*>$ of size $<*>$ from /10.250.10.6" wrongly because there are too many logs reporting data from this IP address and consequently the IP address seems to be a constant part. In the following section, we will show how our vectorization method can mitigate the influence of such errors.

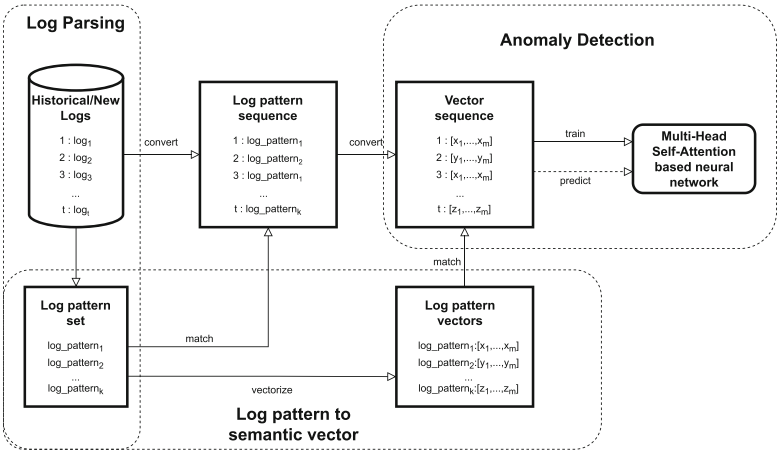

Fig. 2. The framework of LogAttention

3.3 Log Pattern Vectorization

Inspired by semantic vectorization [16], we view the log patterns as natural language and build log pattern semantic vectors to get rid of the dependency on parsing accuracy as much as possible. Different from Template2Vec [12], pre-trained word vectors from FastText [9] are directly used to represent the words in the log pattern. Then, TF-IDF [13] is used to distinguish the different levels of word importance and aggregate these word vectors. Finally, the log pattern vector is calculated by aggregating all the weighted word vectors. In this way, the influence of parsing errors can be minimized because few different words will make little change to the log pattern vector. As shown in Eq. (1), w_i is the TF-IDF score of each word and v_{word_i} is the corresponding word vector.

$$v_{logpattern} = \frac{1}{N} \sum_{i=1}^{N} w_i v_{word_i} \tag{1}$$

The main drawback of TF-IDF in [16] is that IDF is calculated in the log pattern set, where $IDF(word_i) = \frac{L}{L_{word_i}}$, L is the number of all log patterns and L_{word_i} is the number of log patterns containing the target word. In other words, [16] uses the whole log pattern set as the corpus, which is inappropriate. Consider the logs from Hadoop system. There are 48 log patterns extracted by Drain and the word "Exception" occurs in 24 log patterns, which means "Exception" isn't important at all. Obviously, it is not reasonable. Actually, most log patterns containing "Exception" occurs only few times in raw logs, which means "Exception" is very important according to the IDF score in raw logs. To overcome this drawback, we consider the importance of different words in raw logs to better represent the log patterns. In our TF-IDF method, IDF takes the occurrences of log patterns into consideration and TF keeps the same.

The calculation of TF is shown in Eq. (2) where L_{word} is the number of $word_i$ in the log pattern and L_{total} is the number of all words in the log pattern. The calculation of IDF is shown in Eq. (3) where L_{logs} is the number of lines in raw logs and occ_j is the occurrence of $logpattern_j$.

$$TF(word_i) = \frac{L_{word}}{L_{total}} \tag{2}$$

$$IDF(word_i) = \frac{L_{logs}}{\sum_{j=1}^{N} y_j \cdot occ_j}, y_j = \begin{cases} 0 \ word_i \notin logpattern_j \\ 1 \ word_i \in logpattern_j \end{cases} \tag{3}$$

It is noteworthy that our method is still different from directly using TF-IDF in the raw logs because variables are not included in our work. Compared with the TF-IDF used in log pattern set, our implementation reduces the impact of log patterns with the smaller number of occurrences. Combined with the aggregation of word vectors, our vectorization method can reduce the impact of parsing errors as much as possible.

3.4 Log Anomaly Detection

After the step of log pattern vectorization, each log sequence $S = [log_1, \ldots, log_t]$ can be transformed into a matrix $M = [v_{logpattern_1}, v_{logpattern_2}, \ldots, v_{logpattern_k}]$. Taking M as input, LogAttention adopts a self-attention based neural network to detect whether an anomaly is hidden in S.

Self-attention mechanism is a variant of the attention mechanism in which *Query(Q)*, *Key(K)* and *Value(V)* are obtained from the same input. [14] proposes a specific self-attention called multi-head scaled dot-product attention and compares it with recurrent and convolutional layers. The results show that self-attention layers have the better ability to learn long-range dependencies and benefit from the better efficiency. Inspired by these characteristics, we adopt multi-head self-attention layers instead of LSTM to overcome its possible drawbacks in previous work [5,12,16].

The main structure of our proposed model has been shown in Fig. 3. Transformed from M, the input matrix $X \in R^{t \times h}$ can be represented as follows:

$$X = dropout(M^{\top} W^h + PosEmb) \tag{4}$$

$$PosEmb = W^p \tag{5}$$

where $W^h \in R^{m \times h}$ and m is the dimension of original log pattern vector. We use the learned positional embeddings $W^p \in R^{t \times h}$ to represent the absolute position of logs as shown in Eq. (5).

For the i^{th} head, three linear transformations will produce Q_i, K_i and V_i using X:

$$Q_i = XW_i^Q, K_i = XW_i^K, V_i = XW_i^V \tag{6}$$

where $W_i^Q, W_i^K, W_i^V \in R^{h \times d_{head}}$ and $d_{head} = h/heads$.

With Q_i, K_i and V_i in the i_{th} head, scaled dot-product attention mechanism will calculate the attention scores. Then each log pattern embedding will attend

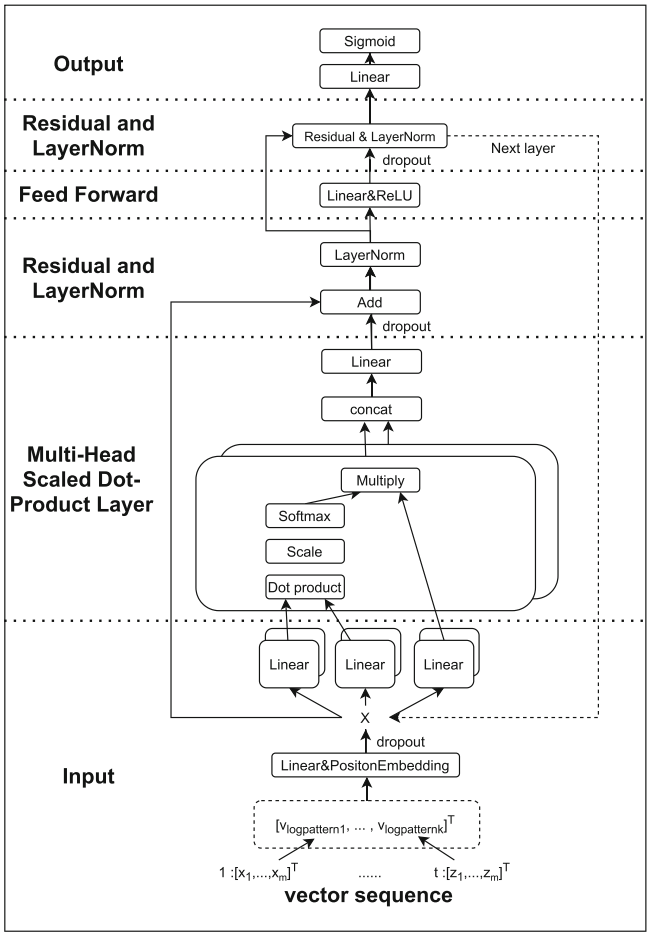

Fig. 3. Multi-head scaled dot-product based neural network

to all the other log patterns within the sequence according to the attention scores. The matrix calculation of scaled dot-product has been shown below:

$$head_i = softmax(\frac{Q_i {K_i}^\top}{\sqrt{d_{head}}})V_i \tag{7}$$

The result of all heads will be concatenated together to produce a new matrix for the next layer by a linear transformation:

$$X_o = (head_1 \oplus head_2 \oplus \cdots \oplus head_i)W^o \tag{8}$$

where $\oplus$ is the operator of concatenation and $W^o \in R^{h \times h}$.

As shown in Eq. (9), two kinds of regularizations will be applied before X^o is fed to the feed-forward layer, including layer normalization and residual dropout.

As shown in Eq. (10), the result of the current loop is then fed to the next loop to perform the same calculation as X.

$$X_o = LN(X + dropout(X_o)) \tag{9}$$

$$X_i = f_{above}(X_{i-1}) \tag{10}$$

After several loops, we get the output $X_{final} \in R^{t \times h}$. We directly use last line of X_{final} denoted as $y^\top$ to calculate the final anomaly probability for the following two reasons: (1) Each element in the sequence has already contained the information of other elements, so has the last one. (2) The last line of X_{final} corresponds to the last log pattern in the log sequence, which is the latest information of the sequence. Therefore, the final anomaly probability is calculated by $y^\top$ as in Eq. (11).

$$AnomalyProb = \sigma(y^\top W^t + b^t) \tag{11}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{12}$$

4 Evaluation

4.1 Datasets and Criteria

Datasets

1. **HDFS**: HDFS represents the Hadoop Distributed File System, which is a distributed Hadoop application [8]. This data set is generated with benchmark workloads and manually labelled as normal or abnormal. Each line of HDFS log contains a unique block ID. Therefore, the operations in logs can be naturally organized by session windows identified by block ID [7]. There are 11,175,629 logs in total and they can be sliced into 575061 sessions, in which 16838 sessions are labelled as anomalous. See [15] for more details. We randomly pick 6000 normal sessions and 6000 abnormal sessions from HDFS as the training set.
2. **BGL**: BGL represents BlueGene/L, which is a supercomputer system at Lawrence Livermore National Labs (LLNL). The logs can be divided into alert messages and non-alert messages, which are identified by alert category tags [8]. There are 4,747,963 logs in total and 348,460 logs are labelled as anomalous. BGL logs have no identifier like "block ID" for each job session. Therefore, we will use sliding windows to slice logs into a set of log sequences. Log sequence is anomalous as long as it contains an anomalous log. We randomly pick 40000 normal log sequences and 40000 abnormal log sequences from BGL as the training set.

Baselines

We compare LogAttention with Logistic Regression (LR) [2], Invariant Mining (IM) [11], PCA [15] and LogRobust [16], which contains supervised approach,

Table 1. Experimental results on HDFS dataset and BGL dataset

	HDFS			BGL		
	Precision	Recall	F1 score	Precision	Recall	F1 score
LogAttention	0.975	0.998	0.986	0.978	0.992	0.985
LogRobust	0.933	0.968	0.950	0.950	0.998	0.974
LR	0.955	0.911	0.933	1	0.810	0.820
PCA	0.980	0.670	0.790	0.500	0.610	0.550
IM	0.880	0.950	0.910	0.830	0.990	0.910

unsupervised approach, log pattern counter based approach and deep learning based approach. All the parameters have been fine-tuned for the best accuracy.

Evaluation Metrics
Following previous work, we use *precision*, *recall* and *F1 score* as the evaluation metrics. Log-based anomaly detection is a binary classification problem. The output can be divided into 4 parts: TP, TN, FP, FN. TP is the number of abnormal sequences classified accurately. TN is the number of normal sequences classified accurately. If an anomalous sequence is wrongly classified as a normal sequence, then it will be viewed as FN. Precision, recall and F1 score are calculated as follows:

$$Precision = TP/(TP + FP) \tag{13}$$

$$Recall = TP/(TP + FN) \tag{14}$$

$$F1score = (2 * Precision * Recall)/(Precsion + Recall) \tag{15}$$

4.2 Overall Results and Analysis

In this section, we compare LogAttention with 4 baselines approaches on two datasets: HDFS and BGL. The implementation of LogAttention on HDFS has 4 layers and 8 heads whose dimensions are 32. The experimental results have been shown in Table 1. It can be clearly seen that LogAttention achieves the best F1 score (0.986) and precision (0.975) on HDFS. The results of Invariant Mining and LogRobust are highly competitive with LogAttention in recall. However, they achieve much lower precision, which means they generate far more false alarms than LogAttention. On the contrary, PCA achieves a high precision of 0.98 but a low recall of 0.67, which means 33% anomalies are ignored.

The implementation of LogAttention on BGL has 2 layers and 8 heads whose dimensions are 4. The implementation of LogRobust has 2 LSTM layers with 32 neurons. The window length has been set to 200 here. It is worth mentioning that the results from any window length can be used as the overall results of BGL, depending on the expected scale. The experimental results have been shown in

Table 1. LogAttention and LogRobust achieve comparable F1 scores of 0.985 and 0.974 respectively, which are obviously higher than other approaches. LogAttention yields a slight degradation of 0.6% in recall, but surpasses LogRobust in precision by a large margin of 2.8%.

In conclusion, the experimental results show that LogAttention performs better than baselines for anomaly detection.

Table 2. Experimental results on BGL dataset with different window lengths (LogAtt: LogAttention; LogRob: LogRobust)

	Window length = 200		Window length = 350		Window length = 450		Window length = 500	
	LogAtt	LogRob	**LogAtt**	LogRob	**LogAtt**	LogRob	**LogAtt**	LogRob
Precision	**0.978**	0.950	0.977	**0.978**	**0.977**	0.966	**0.989**	0.962
Recall	0.992	**0.998**	**0.999**	0.990	**0.999**	0.998	**0.995**	0.994
F1 score	**0.985**	0.974	**0.987**	0.984	**0.988**	0.981	**0.992**	0.978

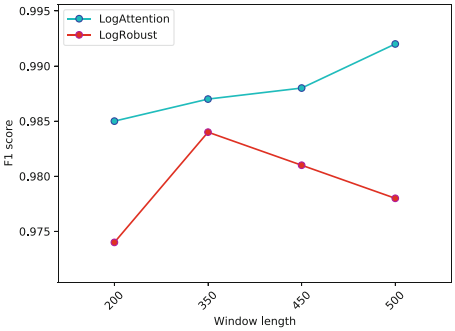

Fig. 4. F1 scores of two approaches with different window lengths

4.3 Experiments on Log Sequences with Different Window Lengths

As described earlier, LogAttention has the ability to learn long-range dependencies in log sequences. To prove this, we evaluate the performance of LogAttention and LogRobust on BGL with different window lengths. LR, PCA and IM will not be considered in this section because they belong to log pattern counter based approaches, which are not sensitive to the parameter of window length and they can't capture sequential patterns in log sequences anyway. The experimental results have been illustrated in Fig. 4, and the more detailed results have been shown in Table 2.

In terms of the F1 score, LogAttention even performs better when the window length becomes larger as shown in Fig. 4. When the window length equals 200, LogAttention achieves the F1 score of 0.985. Then its F1 score increases

continuously by 0.002 and 0.003 respectively when the window length becomes 350 and 450. When the window length equals 500, LogAttention achieves the highest F1 score of 0.992. LogRobust achieves the F1 score of 0.974 when the window length equals 200. Then the F1 score increases by 0.01 when the window length increases to 350. In contrast to LogAttention, the F1 score of LogRobust decreases continuously by 0.003 and 0.006 when the window length increases to 450 and 500. Unlike LogAttention, LogRobust achieves its highest F1 score when the window length is 350. Apart from F1 score, we also zoom in on the metrics of precision and recall. The results are reported in Table 2. It can be seen that LogAttention performs better in most cases. It's obvious that LogAttention is not influenced by the increasing window length and can get better results compared to LogRobust.

Table 3. Experimental results of LogAttention with different vectorization methods

	Metrics	**LogAttention**	LogAttention*
HDFS	Precision	**0.975**	0.962
	Recall	0.998	**0.999**
	F1 score	**0.986**	0.980

4.4 Experiments on Log Pattern Vectorization Method

In this section, we compare our vectorization method with [16]. As the baseline, LogAttention is retrained with the vectors generated by the vectorization method proposed in [16], denoted as LogAttention*. We report the results in precision, recall and F1 score in Table 3. As we can see, LogAttention achieves a precision of 0.975 and a F1 score of 0.986, which is much higher than the retrained LogAttention* by 0.013 and 0.006 respectively. As for the recall, LogAttention* surpasses LogAttention by a small margin of 0.001. The difference is almost indistinguishable, so the performance in recall can be considered as comparable for the two methods. In a word, our vectorization method can lead to better results than the one proposed in [16].

5 Conclusion

In recent years, many deep learning based approaches have been proposed to meet the urgent requirement of log-based anomaly detection. Although these approaches demonstrate their superiority, these approaches still have some drawbacks from the LSTM-based models and the vectorization methods. In this paper, we propose an approach, namely *LogAttention*, to detect anomalies effectively. LogAttention adopts the semantic vectorization method based on TF-IDF and the network based on multi-head scaled dot-product attention. We evaluate

LogAttention on two public available datasets with extensive experiments. The experimental results show that LogAttention can effectively detect anomalies in various cases compared to the existing baselines.

Acknowledgment. This work was supported by National Key R&D Program of China (Grant No. 2020YFB2103300).

References

1. Baril, X., Coustié, O., Mothe, J., Teste, O.: Application performance anomaly detection with LSTM on temporal irregularities in logs. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 1961–1964 (2020)
2. Bodik, P., Goldszmidt, M., Fox, A., Woodard, D.B., Andersen, H.: Fingerprinting the datacenter: automated classification of performance crises. In: Proceedings of the 5th European Conference on Computer Systems, pp. 111–124 (2010)
3. Debnath, B., et al.: LogLens: a real-time log analysis system. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 1052–1062. IEEE (2018)
4. Du, M., Chen, Z., Liu, C., Oak, R., Song, D.: Lifelong anomaly detection through unlearning. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1283–1297 (2019)
5. Du, M., Li, F., Zheng, G., Srikumar, V.: DeepLog: anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1285–1298 (2017)
6. He, P., Zhu, J., Zheng, Z., Lyu, M.R.: Drain: an online log parsing approach with fixed depth tree. In: 2017 IEEE International Conference on Web Services (ICWS), pp. 33–40. IEEE (2017)
7. He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: system log analysis for anomaly detection. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 207–218. IEEE (2016)
8. He, S., Zhu, J., He, P., Lyu, M.R.: Loghub: a large collection of system log datasets towards automated log analytics. arXiv preprint arXiv:2008.06448 (2020)
9. Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., Mikolov, T.: Fasttext.zip: compressing text classification models. arXiv preprint arXiv:1612.03651 (2016)
10. Liang, Y., Zhang, Y., Xiong, H., Sahoo, R.: Failure prediction in IBM BlueGene/l event logs. In: Seventh IEEE International Conference on Data Mining (ICDM 2007), pp. 583–588. IEEE (2007)
11. Lou, J.G., Fu, Q., Yang, S., Xu, Y., Li, J.: Mining invariants from console logs for system problem detection. In: USENIX Annual Technical Conference, pp. 1–14 (2010)
12. Meng, W., et al.: LogAnomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: IJCAI, pp. 4739–4745 (2019)
13. Salton, G., Buckley, C.: Term-weighting approaches in automatic text retrieval. Inf. Process. Manage. **24**(5), 513–523 (1988)
14. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
15. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.I.: Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, pp. 117–132 (2009)

16. Zhang, X., et al.: Robust log-based anomaly detection on unstable log data. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 807–817 (2019)
17. Zhu, J., et al.: Tools and benchmarks for automated log parsing. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 121–130. IEEE (2019)