LogAttn: Unsupervised Log Anomaly Detection with an AutoEncoder Based Attention Mechanism

Linming Zhang[1,2], Wenzhong Li[1(✉)], Zhijie Zhang[1], Qingning Lu[1], Ce Hou[1], Peng Hu[2], Tong Gui[2], and Sanglu Lu[1]

[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
lwz@nju.edu.cn

[2] Huawei Nanjing Research Center, Nanjing 210012, China

Abstract. System logs produced by modern computer systems are valuable resources for detecting anomalies, debugging performance issues, and recovering application failures. With the increasing scale and complexity of the log data, manual log inspection is infeasible and man-power expensive. In this paper, we proposed LogAttn, an autoencoder model that combines an encoder-decoder structure with an attention mechanism for unsupervised log anomaly detection. The unstructured normal log data is proceeded by a log parser that uses a semantic analyse and clustering algorithm to parse log data into a sequence of event count vectors and semantic vectors. The encoder combines deep neural networks with an attention mechanism that learns the weights of different features to form a latent feature representation, which is further used by a decoder to reconstruct the log event sequence. If the reconstruction error is above a predefined threshold, it detects an anomaly in the log sequence and reports the result to the administrator. We conduct extensive experiments based on three real-world log datasets, which show that LogAttn achieves the best comprehensive performance compared to the state-of-the-art methods.

Keywords: Log anomaly detection · Sematic feature · AutoEncoder · Attention mechanism · Unsupervised learning

1 Introduction

System logs are universally produced by modern computer system to record operation states and critical events, which are valuable resources for detecting system anomalies, debugging performance issues, and recovering application failures [10,14,17]. With the increasing scale and complexity of computer systems, the number of logs could be millions, bringing the challenges for the administrators to fully understand the system status and detect anomalies efficiently. While manual log inspection is infeasible and man-power expensive, automated log anomaly detection has become an urgent task and a valuable research topic [24,28].

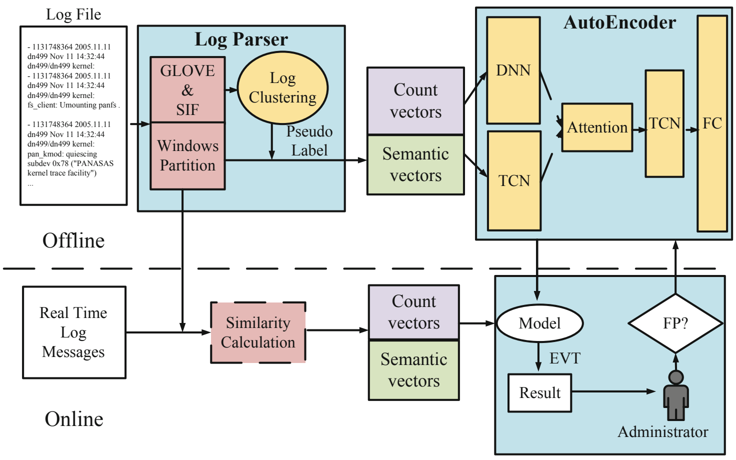
In the recent years, many methods of automatic log anomaly detection have been proposed to mine abnormal events in logs through machine learning techniques [4,18,19]. Xu [28] proposed to detect anomalies in the log with a principal component analysis (PCA) approach. Lou [20] mined invariants in the log, where the change of invariants is considered as an anomaly in the system. Min et al. proposed DeepLog [6], deep model based on long short-term memory (LSTM) network to detect anomalies by analyzing the temporal information of the log sequence.

Despite that learning-based log anomaly detection has made great progress, it still confronts the following challenges. (1) Unstructured log parsing. Most log data are unstructured and they requires a parsing method to map unstructured log into structured format for data mining. Existing methods [27] [13] parse unstructured log into predefined log template, which require expert knowledge to define parsing rules and they are not extensible to new log events [16]. (2) Semantic feature extraction. Log data contain rich semantic information that are useful for understanding log anomalies. Previous works [6] intended to detect abnormal execution sequence from log data, and the semantic features has not been fully addressed. (3) Unsupervised anomaly detection. Since annotating log anomalies is complicated and man-power expensive, most log data are unlabeled. In practice, the detection of log anomalies is expected to be proceeded in an unsupervised way.

In order to address the above challenges, we proposed LogAttn, a log anomaly detection framework with an autoencoder based attention mechanism. We use deep learning method to embed log events and semantic features into latent vectors, and apply a clustering method to parse unstructured log data into pseudo labels. We propose an autoencoder model that combines an encoder-decoder structure with an attention mechanism for unsupervised log anomaly detection. The encoder uses an temporal convolutional network (TCN) to capture temporal semantic correlations and a deep neural network (DNN) to capture statistical correlations. The attention mechanism learns the weights of different features to form a latent feature representation, which is further used by a decoder to reconstruct the log event sequence. The overall model is trained with normal log data, and then executed online to detect anomaly events by comparing the reconstruction error with a predefined threshold. We conduct extensive experiments based on three real-world open system log datasets, which show that LogAttn has a better trade-off between precision and recall, and achieve the best comprehensive performance compared to the state-of-the-art methods.

2 Related Work

Log anomaly detection methods can be divided into two categories: supervised and unsupervised. Supervised log anomaly detection methods include SVM [12], LR [18], LOGROBUST [29], Decision Tree [5], etc. Due to the difficulties of annotating log data in reality, this paper only focuses on unsupervised log anomaly detection.


```
- 1131748364 2005.11.11
dn499 Nov 11 14:32:44
dn499/dn499 kernel:
- 1131748364 2005.11.11
dn499 Nov 11 14:32:44
dn499/dn499 kernel:
fs_client: Umounting panfs .

- 1131748364 2005.11.11
dn499 Nov 11 14:32:44
dn499/dn499 kernel:
pan_kmod: quiescing
subdev 0x78 ("PANASAS
kernel trace facility")
...
```

Fig. 1. The architecture of LogAttn.

Unsupervised log anomaly detection can be further divided into shallow learning based methods and deep learning based methods. Conventional unsupervised machine learning techniques were applied for log anomaly detection, which included PCA [28], IM [20]. Xu et al. [28] applied Principal Component Analysis (PCA) to generate the log normal and abnormal subspace, where the normal subspace is the first K principal components and the abnormal subspace is the rest of the dimensions. Lou et al. [20] applied an invariant mining (IM) algorithm to find the linear relationship maintained by the system under different inputs or loads. If any invariants are broken, the log sequence is considered an anomaly. Recently, deep learning techniques were introduced to unsupervised log anomaly detection. LogCluster [19] was a clustering based log anomaly detection method that groups similar log sequences by clustering them and detects anomaly if the nearest group is abnormal. Deeplog [6] was a deep learning based unsupervised log anomaly detection method that used Long Short-term Memory (LSTM) to detect abnormal log sequences. LogAnomaly [22] adopted a Template2Vec method to extract the semantic information hidden in the log template, and detected both continuous and quantitative log anomaly using a neural network.

Different from the existing unsupervised log anomaly detection methods that relied on predefined log templates, our work uses word embedding and clustering method to form log representations, and adopts a deep generative model combining with an attention mechanism for unsupervised log anomaly detection.

3 LogAttn Mechanism

3.1 Framework

We proposed an unsupervised framework called LogAttn for system log anomaly detection, which is illustrated in Fig. 1. The overall framework is divided into two parts: offline training and online detection.

During offline training, the unstructured normal log data is proceeded by a *log parser* to form formalized representations, which are used to train an *autoencoder (AE) model* to learn the normal execution pattern. The log parser uses a semantic analyse and clustering algorithm to parse log data into a sequence of event count vectors and semantic vectors. The autoencoder model is an encoder-decoder structure with an attention mechanism. The encoder uses an temporal convolutional network (TCN) to capture temporal semantic correlations and a deep neural network (DNN) to capture statistical correlations. The hidden layer of the encoder is connected to an *attention layer* to learn the weights of different features to form a latent feature representation, which is further used by a decoder to reconstruct the log event sequence. By training the AE model in an unsupervised way, it can reconstruct the normal log execution sequence effectively.

During online execution, the newly generated log sequence is proceeded by the log parser, and then fed to the well-trained AE model to reconstruct the input sequence. The error between the reconstructed log sequence and the original log sequence is calculated. If the log follows a normal pattern, it should be successfully reconstructed with very low error. If the reconstruction error is above a predefined threshold, it detects an anomaly in the log sequence and reports the result to the administrator.

3.2 Log Parsing Based on Semantic Analysis and Clustering

Log parsing is the first and indispensable part of log anomaly detection, which converts the log's text messages into a sequence of execution flows. Unlike traditional log parsers (such as LKE [9], LogSig [27], Drain [13], IPLoM [21], etc.) that used predefined templates to parse text messages, we propose a novel log parsing method that uses semantic analysis to embed log events into latent vectors, and applies clustering on the latent vectors to form their pseudo labels (represented by cluster IDs). The detailed process is described as follows.

(1) We first apply word embedding on the log sentence through Global Vectors for Word Representation (GloVe) [25]. GloVe is a word representation tool based on global word frequency statistics, which can convert a word into an embedding vector representation that captures semantic features among words, such as their similarity, analogy and so on.
(2) We then calculate the weighted coefficient of each word using Smooth Inverse Frequency (SIF) [1] to form sentence vectors. The lower the frequency of a word appears in a sentence, the more important it is in the sentence, corresponding to a larger weighted coefficient. Thereafter, we let each sentence vector subtracts its projection on the first principal component of the matrix composed of all sentence vectors, which can erase the common information of all sentences and increase the discrimination among the sentence embedding vectors.
(3) After obtaining the embedding vectors of the log events, we cluster them through the DBSCAN [8] algorithm, and use the cluster IDs to form the

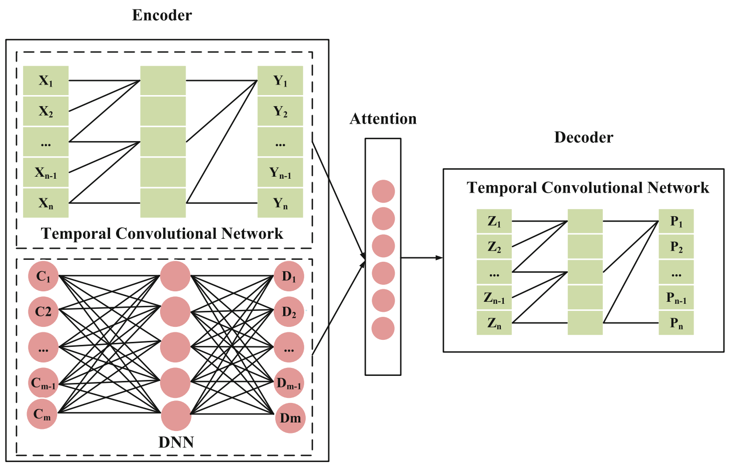


Fig. 2. The structure of encoder-decoder with an attention mechanism.

pseudo labels of the log events. By substituting the log events with the cluster IDs, we can parse the log text as a sequence of execution flow represented by pseudo labels.

(4) We further adopt a window-based partition to divide the log data into subsequences. A subsequence is a basic unit in our model to detect whether there is an anomaly. The log partition can be carried out with a sliding window or a session window. The sliding window partitions log data along time steps into overlapping subsequences (with moving forward a step size). The session window partitions log data based on session IDs where a unique session ID corresponds to a subsequence representing an execution flow of a session.

(5) Based on window partition and the embedding vector of each sentence, we form a sequence of *semantic vectors* for each log subsequence represented by the embedding vectors of the log events.

(6) Based on the pseudo labels and window partition, we form a sequence of *count vectors* for the log subsequences. A count vector is formed by calculating the occurrence frequency of each class of event in the log sequence, which represents the statistical pattern of a log subsequence.

3.3 Log Anomaly Detection Model

The proposed log anomaly detection model is illustrated in Fig. 2, which is the combination of an autoencoder and an attentional mechanism. The encoder uses a temporal convolutional network (TCN) to process the semantic vector sequence and a deep neural network (DNN) to process the corresponding event count sequence to form compact representations of both temporal and sematic features. An attention mechanism is assigned to learn the importance weights of sematic and statistical features. The decoder takes both attention weights and compact feature vectors as input to reconstruct the subsequence of log events. The overall model is trained offline with normal log data, and then executed online to detect anomaly events by comparing the reconstruction error with a predefined threshold. The detail process is explained in the following.

AutoEncoder. Anomaly detection based on dimensionality reduction assumes that the data have a certain degree of correlation and can be embedded into a lower latitude subspace [30]. After the original data is embedded in a lower latitude, the abnormal and normal data will be separated. Autoencoder is a powerful unsupervised learning technique for information compression, where an encoder is used to find a compressed representation of a given data, while a decoder is used to reconstruct the original input. During training, the decoder forces the encoder to select the feature with the most useful information, which is preserved in the compressed representation.

The encoder we used for log data contains a temporal convolutional network (TCN) [3]and a deep neural network (DNN), which are explained in the following.

The time convolution network (TCN) is a new type of neural network model derived from convolutional neural network (CNN). Unlike normal convolution, TCN uses the causal convolution and void convolution to extract features across time steps, which is powerful to capture the temporal dependencies of sequence data and provide a visual field for temporal modeling. In TCN, dilated causal convolutions are used to allow the filter to be applied to a region larger than the length of the filter itself by skipping part of the input, which are formulated by

$$a_{l,t} = \sum_{i=0}^{k_l^a - 1} f_l^a(i) \cdot a^a_{l-1,t-d_l^a \cdot i}, \tag{1}$$

where $a_{l,t}$ is the output of layer l at time t, and f_l^a, k_l^a, d_l^a are the filter, filter size and dilation factor of the layer respectively.

The deep neural network (DNN) used in our model is a three-layered fully-connected neural network, whose structure is illustrated in the left part of Fig. 2. It is used to capture the statistical characteristics of log event count sequence.

Attention Mechanism. The latent representations of the semantic features and statistical features generated from the encoder are connected to an attention layer [2] to fuse the heterogeneous features and learn the importance of different elements in the feature vectors. Therefore, the attention mechanism can be seen as an interface between the encoder and decoder, providing the decoder with the importance weights from the hidden stats. With this setup, the model can selectively focus on the useful parts of the input sequence to learn the "alignment" between them.

At time step t, denote the encoder's output by a vector of length s with elements Y_t, Y_{t-1}, ..., Y_{t-s+1}. The attention mechanism can be formally represented by

$$Z_t = \sum_{k=0}^{s-1} \alpha_{t,k} Y_{t-k} \tag{2}$$

$$\alpha_{t,k} = \frac{\exp\left(e_{t,k}\right)}{\sum_{k=0}^{S-1} \exp\left(e_{t,k}\right)} \tag{3}$$

$$e_{t,k} = P\left(Y_{t-k}\right) = v^T \tanh\left(W Y_{t-k} + b\right) \tag{4}$$

where $e_{t,k}$ is the importance of Y_{t-k}; $a_{t,k}$ is the normalized value of the importance; and W, b and v are the model's parameters to be learned.

Loss Function. The decoder consists of a TCN network that has the same structure as the encoder, which is normalized by a softmax layer to reconstruct the input log subsequence. Multiple vectors in the reconstructed sequence represent the probability of the possible types of log events at the current time. We use *cross entropy* [11] as the loss function of the decoder, which is given by

$$\text{Loss} = -\sum_{i=1}^{l} \sum_{j=1}^{N} R_{i,j} \log P_{i,j}, \tag{5}$$

where l is the length of the log subsequence and N is the number of log events; $R_{i,j} \in \{0, 1\}$ is an indicator, i.e., $R_{i,j} = 1$ if the i-th log event belongs to the j-th pseudo label, and $R_{i,j} = 0$ otherwise; and $P_{i,j}$ is the output of the decoder representing the probability that the i-th log event belongs to the j-th pseudo label.

3.4 Selection of Anomaly Threshold

After training the autoencoder, the reconstructed sequence of the log can be obtained, and the *reconstruction error* between the reconstructed sequence and the original sequence can be quantified using the Kullback–Leibler (KL) divergence [7], which is calculated by

$$D_{KL}(p\|q) = \sum_{i=1}^{N} \left[p\left(x_i\right) \log p\left(x_i\right) - p\left(x_i\right) \log q\left(x_i\right)\right], \tag{6}$$

where $p(\cdot)$ and $q(\cdot)$ are the reconstructed log events and the original log events accordingly, and N is the total number of log events in the subsequence.

If the reconstruction error exceeds a threshold, an anomaly is considered to be detected. Here we use the extreme value theory (EVT) [26] to derive the anomaly threshold. The goal of extreme value theory is to find the law of extreme events, which is generally considered to be different from the distribution of the whole data. Extreme value theory makes it possible to detect those extreme events without considering the complex distribution of the original data.The peak-over-threshold (POT) [26] in the extreme value theory uses Generalized Pareto distribution (GPD) to fit the extreme value beyond the threshold. We use the POT to learn the threshold of anomaly log with the following formulas.

$$\bar{F}_{\tau}(x) = \mathbb{P}(\tau - X > x \mid X < \tau) \underset{\tau \to \tau}{\sim} \left(1 + \frac{\gamma x}{\sigma(\tau)}\right)^{-\frac{1}{\gamma}}, \tag{7}$$

where σ and γ are the parameters of GPD, τ is the anomaly threshold, X is the difference value, and $\tau - X$ represents the part beyond the threshold τ.

The parameters δ and γ are estimated using maximum likelihood estimation (MLE). The estimated values $\hat{\sigma}$ and $\hat{\gamma}$ are used to calculate the quantile under a given anomaly probability q,

$$z_q \simeq \tau - \frac{\hat{\sigma}}{\hat{\gamma}}\left(\left(\frac{qn}{N_{\tau}}\right)^{-\hat{\gamma}} - 1\right), \tag{8}$$

where τ is the empirical threshold of anomaly detection, n is the total number of all observed values, and N_{τ} is the number of points less than τ.

4 Performance Evaluation

4.1 Experimental Environment

Implementation. We conduct experiments on a personal computer (CPU: Intel Core i7-8900U 1,8 GHz, Memory: 16 GB DDR4 2666 MHz, and OS: 64-bit Ubuntu 16.04). The default parameter settings are as follows. The TCN structure of our encoder and decoder are the same (a three-layer TCN network), and the DNN in our encoder is a three-layer fully connected network. The window size of the log sequence is 10. The anomalous proportion of EVT is set to 0.08.

Datasets. We use three open system log datasets to evaluate the algorithms, which are described in the following.

- **HDFS** [28]: The HDFS dataset was generated from a Hadoop cluster consisting of 200 amazon EC2 nodes. There were 11,197,954 log entries in the dataset, of which anomaly accounted for 2.9%. We split the log entries into different sessions using the identity character block_id. These sessions are flagged by Hadoop domain experts. We used 4,855 sessions as the training set, which was parsed from the previous 100,000 log entries. The remaining 553,366 normal sessions and 15,838 abnormal sessions were used as the test set.
- **BGL** [23]: The BGL dataset was collected from a Blue Gene/L supercomputer system deployed on Lawrence Livermore National Labs (LLNL). The dataset contains a total of 4,747,963 log entries, of which 348,460 are marked as anomaly. The BGL dataset uses the first 80% (based on the log timestamp) as the training set and the last 20% as the test set.

– **Thunderbird** [23]: The Thunderbird dataset was collected from a Thunderbird supercomputer system at Sandia National Labs (SNL) in Albuquerque, with 9,024 processors and 27,072GB of memory. The first four million log entries in the dataset were analyzed, with the first 80% (based on the timestamp of the log) as the training set and the last 20% as the test set.

Performence Metrics. We use the following standard performance metrics to evaluate the algorithms: Precision (P), Recall (R), and F1-score (F_1), given by

$$P = \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN}, \quad F_1 = 2 \times \frac{P \times R}{P + R}, \tag{9}$$

where TP is the True Positives, FP is the False Positives, and FN is the False Negatives.

Baseline Algorithms. We compare the proposed LogAttn algorithm with five unsupervised log anomaly detection methods: two shallow learning based methods *PCA* [28] and *IM* [20], and three deep learning based methods *LogCluster* [19], *Deeplog* [6], and *LogAnomaly* [22]. We implement LogAttn and DeepLog with the deep learning python library PyTorch. For PCA, IM, and LogCluster, we use their open source toolkit [15].

Table 1. Performance of log anomaly detection algorithms on different datasets (P means Precision; R means Recall; F1 means F1-score; '-' means unavailable)

Methods	HDFS			BGL			Thunderbird		
	P	R	F1	P	R	F1	P	R	F1
PCA	**0.98**	0.74	0.79	0.50	0.61	0.55	0.51	0.55	0.53
LogCluster	0.87	0.74	0.80	0.42	0.87	0.57	0.48	0.50	0.49
IM	0.88	0.95	0.91	0.83	**0.99**	0.91	0.73	0.81	0.77
DeepLog	0.95	0.96	0.96	0.90	0.95	0.93	0.88	0.92	0.90
LogAnomaly	0.97	0.94	0.96	**0.96**	0.94	0.95	-	-	-
LogAttn	0.95	**0.99**	**0.97**	**0.96**	0.98	**0.97**	**0.91**	**0.93**	**0.92**

4.2 Numerical Results

We perform comparative experiments between LogAttn and baselines on the above datasets. Table 1 are the experimental results on HDFS, BGL, and Thunderbird respectively. In general, LogAttn performed well in all three datasets, reaching the F1-score of 0.97,0.97 and 0.91.

As can be seen from the experimental results, LogAttn, DeepLog and IM are the most effective log anomaly detection methods, while LogAttn has the best overall performance, reaching the F1-score of 0.97, 0.97, and 0.91 accordingly

Table 2. Performance of LogAttn with/without semantic analysis on BGL dataset (P means Precision; R means Recall; F1 means F1-score).

Template	w/o semantic analysis			w/ semantic analysis		
	P	R	F1	P	R	F1
T1	0.94	0.98	0.96	0.96	0.99	0.98
T2	0.90	0.98	0.94	0.96	0.98	0.97
T3	0.88	0.99	0.93	0.95	0.98	0.96

on the three datasets. PCA and LogCluster have poor performance, and their F1-scores on the three datasets are lower than 0.80.

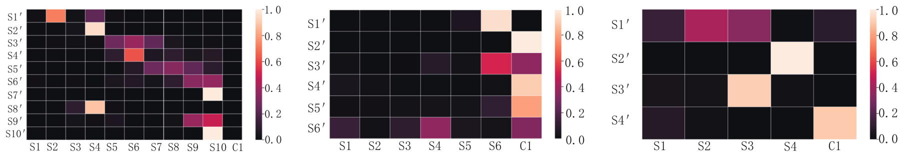
In HDFS dataset, IM is less accurate, but has a recall rate of 0.95. DeepLog also had a recall rate of 0.96, but an accuracy rate of 0.95. LogAttn significantly improved the recall rate to 0.99 and F1-score to 0.97 while ensuring a high accuracy. The dramatic increase in recall rate means that almost all anomalies can be detected by LogAttn automatically, leading to a large saving of manpower, time and resources.

In the BGL dataset, IM and DeepLog all have good comprehensive performance. The recall rate of IM is as high as 0.99, but unfortunately the precision is only 0.83. This unbalanced experimental result means that although almost all the anomalies are detected, there are large amount of normal logs detected as anomalies. DeepLog also has a recall rate of 0.96 with an accuracy of 0.90, which is obviously a better trade-off than the other algorithms.

In the Thunderbird dataset, the experimental results are similar. The comprehensive performance of IM, DeepLog and LogAttn is above 0.80, while that of PCA and LogCluster is poor. The recall rate of DeepLog and LogAttn was 0.92, but the precision and F1-score of LogAttn are higher than that of DeepLog.

Ablation Study of Semantic Features. To show that the addition of semantic analysis can improve the robustness of the whole system, we conduct comparative experiments on the basis of three different number of template libraries T1, T2, T3 of BGL respectively, and the results of with/without semantic analysis are recorded. As shown in Table 2, with the same number of templates, log anomaly detection tasks perform better with semantic analysis. The difference in the number of templates will affect the experimental results. We can see that the comprehensive performance of three different templates stabilized above 0.96 in the case of semantic analysis, while the F1-score without semantic analysis 0.96, 0.94, 0.93 for T1, T2, T3 accordingly. With the same template, the performance of semantic analysis is better than that of non-semantic analysis. We can also see from Table 2 that the change in the number of templates has a small impact on performance when there is semantic analysis, while the performance fluctuation is larges when there is no semantic analysis.

Visualization of Attention Weights. Both semantic information and statistical information have different priorities for log anomaly detection, which are represented by their attention weights. The attention mechanism enables automatic selection and fusion of log characteristics. In order to evaluate the impact of the combination of semantic and statistical characteristics of the subsequence on the log anomaly detection task, we perform a visualization analysis on the attention weight for the HDFS, BGL, and Thunderbird dataset.



(a) HDFS. (b) BGL. (c) Thunderbird.

Fig. 3. Visualization of attention weights. It is shown that HDFS has larger weights on semantic features; BGL has larger weights on statistical features; Thunderbird has large weights on both.

As shown in Fig. 3(a), S1-S10 is the semantic vector sequences, and C1 is the count vector sequence. The concatenation of S1-S10 and C1 is taken as the input of the attention model, and S1′-S10′ is the output. From the results, we can see that parts of the semantic features have much higher weights than the rest features. As shown in Fig. 3(b), S1-S6 is the sequential sequence of logs, and C1 is the quantitative sequence of logs, which shows that the statistical information in BGL have higher weights than that of the semantic information. Similar conclusion is found in Thunderbird (Fig. 3(c)).

5 Conclusion

Log anomaly detection is a valuable research topic for modern computer systems to debug performance issues and recover application failures. In this paper, we proposed an autoencoder model called LogAttn that combined an encoder-decoder structure with an attention mechanism for unsupervised log anomaly detection. It developed a log parser that used a semantic analyse and clustering algorithm to parse log data into a sequence of event count vectors and semantic vectors. The encoder combined neural networks with an attention mechanism to learn the weights of different features to form a latent feature representation, which was used by the decoder to reconstruct the log event sequence. If the reconstruction error was above a predefined threshold, an anomaly in the log sequence was detected and reported to the administrator. Extensive experiments based on three open log datasets showed that LogAttn outperformed the state-of-the-art methods.

Acknowledgment. This work was partially supported by the National Key R&D Program of China (Grant No. 2018YFB1004704), the National Natural Science Foundation of China (Grant Nos. 61972196, 61832008, 61832005), the Key R&D Program of Jiangsu Province, China (Grant No. BE2018116), the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Sino-German Institutes of Social Computing.

References

1. Arora, S., Liang, Y., Ma, T.: A simple but tough-to-beat baseline for sentence embeddings. In: International Conference on Learning Representations (ICLR 2017) (2017)
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)
3. Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271 (2018)
4. Breier, J., Branišová, J.: Anomaly detection from log files using data mining techniques. In: Kim, Kuinam J. (ed.) Information Science and Applications. LNEE, vol. 339, pp. 449–457. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46578-3_53
5. Chen, M., Zheng, A.X., Lloyd, J., Jordan, M.I., Brewer, E.: Failure diagnosis using decision trees. In: International Conference on Autonomic Computing (2004)
6. 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)
7. Erven, T., Harremoës, P.: Rényi divergence and kullback-leibler divergence. IEEE Trans. Inf. Theory **60**, 3797–3820 (2014)
8. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD 1996) (1996)
9. Fu, Q., Lou, J.G., Wang, Y., Li, J.: Execution anomaly detection in distributed systems through unstructured log analysis. In: IEEE international conference on data mining (ICDM 2009), pp. 149–158. IEEE (2009)
10. Gai, K., Qiu, M., Zhao, H., Sun, X.: Resource management in sustainable cyber-physical systems using heterogeneous cloud computing. IEEE Trans. Sustain. Comput. **3**, 60–72 (2018)
11. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, Massachusetts (2011)
12. He, P., Zhu, J., He, S., Li, J., Lyu, M.R.: Towards automated log parsing for large-scale log data analysis. IEEE Trans. Dependable Secure Comput. **15**(6), 931–944 (2017)
13. He, P., Zhu, J., Zheng, Z., Lyu, M.R.: Drain: an online log parsing approach with fixed depth tree. In: IEEE International Conference on Web Services, pp. 33–40. IEEE (2017)

14. He, S., Lin, Q., Lou, J.G., Zhang, H., Lyu, M.R., Zhang, D.: Identifying impactful service system problems via log analysis. In: 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 60–70 (2018)
15. He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: system log analysis for anomaly detection. In: IEEE 27th International Symposium on Software Reliability Engineering (ISSRE 2016), pp. 207–218. IEEE (2016)
16. Kabinna, S., Bezemer, C.-P., Shang, W., Syer, M.D., Hassan, A.E.: Examining the stability of logging statements. Empir. Softw. Eng. **23**(1), 290–333 (2017). https://doi.org/10.1007/s10664-017-9518-0
17. Khatuya, S., Ganguly, N., Basak, J., Bharde, M., Mitra, B.: Adele: anomaly detection from event log empiricism. In: IEEE Conference on Computer Communications (INFOCOM 2018), pp. 2114–2122. IEEE (2018)
18. Liang, Y., Zhang, Y., Xiong, H., Sahoo, R.: Failure prediction in ibm bluegene/l event logs. In: IEEE International Conference on Data Mining (ICDM 2007), pp. 583–588. IEEE (2007)
19. Lin, Q., Zhang, H., Lou, J.G., Zhang, Y., Chen, X.: Log clustering based problem identification for online service systems. In: IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C 2016), pp. 102–111. IEEE (2016)
20. 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 (ATC 2010), pp. 1–14 (2010)
21. Makanju, A.A., Zincir-Heywood, A.N., Milios, E.E.: Clustering event logs using iterative partitioning. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2009), pp. 1255–1264 (2009)
22. Meng, W., Liu, Y., Zhu, Y., Zhang, S., Zhou, R.: Loganomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI 2019) (2019)
23. Oliner, A., Stearley, J.: What supercomputers say: a study of five system logs. In: IEEE/IFIP 37th International Conference on Dependable Systems and Networks (DSN 2007), pp. 575–584. IEEE (2007)
24. Pecchia, A., Cotroneo, D., Kalbarczyk, Z., Iyer, R.K.: Improving log-based field failure data analysis of multi-node computing systems. In: IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN 2011), pp. 97–108. IEEE (2011)
25. Pennington, J., Socher, R., Manning, C.D.: Glove: global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP 2014), pp. 1532–1543 (2014)
26. Siffer, A., Fouque, P.A., Termier, A., Largouët, C.: Anomaly detection in streams with extreme value theory. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2017) (2017)
27. Tang, L., Li, T., Perng, C.S.: LogSig: generating system events from raw textual logs. In: Proceedings of the 20th ACM international conference on Information and knowledge management (CIKM 2011), pp. 785–794 (2011)
28. 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)

29. Zhang, X., Li, Z., Chen, J., He, X., Cheng, Q.: Robust log-based anomaly detection on unstable log data. In: Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (2019)
30. Zhou, C., Paffenroth, R.C.: Anomaly detection with robust deep autoencoders. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2017), pp. 665–674 (2017)