LogGAN: A Sequence-Based Generative Adversarial Network for Anomaly Detection Based on System Logs

Bin Xia[1], Junjie Yin[1], Jian Xu[2], and Yun Li[1](✉)

[1] Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing, China
{bxia,liyun}@njupt.edu.cn

[2] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China
dolphin.xu@njust.edu.cn

Abstract. System logs which trace system states and record valuable events comprise a significant component of any computer system in our daily life. There exist abundant information (i.e., normal and abnormal instances) involved in logs which assist administrators in diagnosing and maintaining the operation of the system. If diverse and complex anomalies (i.e., bugs and failures) cannot be detected and eliminated efficiently, the running workflows and transactions, even the system, would break down. Therefore, anomaly detection has become increasingly significant and attracted a lot of research attention. However, current approaches concentrate on the anomaly detection in a high-level granularity of logs (i.e., session) instead of detecting log-level anomalies which weakens the efficiency of responding anomalies and the diagnosis of system failures. To overcome the limitation, we propose a sequence-based generative adversarial network for anomaly detection based on system logs named LogGAN which detects log-level anomalies based on the patterns (i.e., the combination of latest logs). In addition, the generative adversarial network-based model relieves the effect of imbalance between normal and abnormal instances to improve the performance of capturing anomalies. To evaluate LogGAN, we conduct extensive experiments on two real-world datasets, and the experimental results show the effectiveness of our proposed approach to log-level anomaly detection.

Keywords: Anomaly detection · Generative adversarial network · Log-level anomaly · Negative sampling

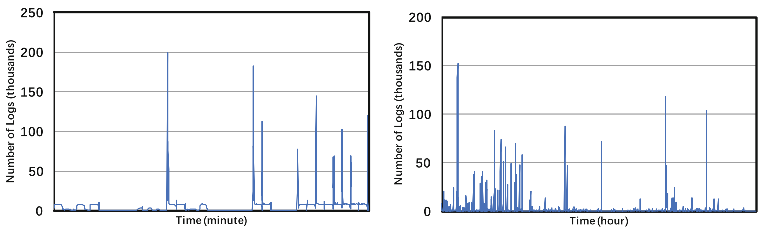
1 Introduction

Anomaly detection is an important task in protecting our daily life from those intended or unintended malicious attacks such as the network intrusion, mobile fraud, industrial damage, and abnormal condition of system [3]. However, with

F. Liu et al. (Eds.): SciSec 2019, LNCS 11933, pp. 61–76, 2019.
https://doi.org/10.1007/978-3-030-34637-9_5

the rapid development of computer science, systems and applications become increasingly complex which makes anomalies diverse and non-trivial to be detected even by human beings. Except for the intended malicious attacks, unknown bugs and errors which are seemingly controllable but caused by non-artificial reason in online systems damage the secure and reliable operating environment. Therefore, the effectiveness and efficiency of anomaly detection have become a big challenge for the further development of information-based society.

Currently, the automated generation of logs is an indispensable component of any large scale system. System logs trace every status of the system and record each critical event in detail to assist administrators in diagnosing bugs, failures, and errors of systems. Therefore, the density of arrival logs and the description of logs directly determine the value of the quantity of knowledge for improving the performance of running systems [9,15]. For example, if arrival logs are extremely dense, it is a challenge to analyze the dependency between events due to the concurrency of logs. Likewise, if the description of logs is colloquial and obscure to represent the state of a system, it is non-trivial to trace the workflows. Figure 1 illustrates the arrival frequency of system logs in practical scenarios, where Fig. 1a shows the logs generated by 203 nodes during 2 days in HDFS and Fig. 1b illustrates the logs generated by 1 node during 215 days in BlusGene/L. Observed from Fig. 1, the peak frequency of arrival logs is 198,878/min and 152,929/hour for HDFS and BGL, respectively. In addition, the number of normal instances is much more than that of anomalies, and generally, anomalies are unlabeled. Therefore, such an extremely frequent arrive of unlabeled logs results in a significant challenge to the prompt response and the precise diagnosis.



(a) HDFS (b) BlusGene/L

Fig. 1. Arrival frequency of system logs in the real-world datasets

To overcome the challenges mentioned above, researchers take a lot of efforts on the anomaly detection based on system logs. The proposed approaches are mainly categorized into the supervised, semi-supervised, and unsupervised strategy based on the availability of labeled data (i.e., normal and abnormal instances). Most of these approaches have good performance in detecting anomalies based on diverse system logs. However, there exist two problems in restricting the further development of system diagnosis [1,11–13]. First, these approaches detect session-level anomalies where a session contains many logs and is divided base on some rules (e.g., period, transaction, and node). In other words, the session including abnormal logs will be detected, however, the abnormal logs cannot be located in the session. Therefore, administrators need to diagnose the workflows in the session which is a non-trivial task. Second, the anomaly is not alerted until the logs are traversed in the session. In other words, the anomaly cannot be detected and responded efficiently when the abnormal log is appearing. This two problems significantly limit the effectiveness and efficiency of system diagnosis.

In this paper, we cast the task of anomaly detection as a pattern-based sequential prediction and propose an LSTM-based generative adversarial network to distinguishing upcoming abnormal events named LogGAN based on temporal system logs. First, we exploit a customized log parser to extracting the structured information (i.e., timestamps, signature, and parameters) and transforming each log into an event. Second, the combinations of events (i.e., pattern) and the corresponding upcoming event are collected from temporal system logs using the sliding window. The collected pairs of patterns and events are utilized to construct real training dataset. LogGAN consists of two major components: (1) generator and (2) discriminator. The generator tries to capture the distribution of real training dataset and synthesizes plausible instances (i.e., normal and abnormal data), while the discriminator aims to distinguish the fake ones from the dataset which is built using the real and synthetic data. Finally, the fully-trained generator is applied to detect whether the upcoming log is normal or abnormal based on the latest events. According to the game setting of anomaly detection, the problem of the imbalance between normal and abnormal instances can be relieved by generating 'real' anomalies to supply the real anomalies in the training set. In addition, the LSTM-based generator identifies whether each upcoming log is normal or abnormal, which efficiently responds alerts of anomalies and effectively assists administrators to diagnose workflows, instead of detecting abnormal sessions including anomalies. To the best our knowledge, this is the first attempt to apply a game setting (i.e., adversarial learning) for the anomaly detection based on system logs. Our contribution can be summarized as below:

- A generative adversarial network is proposed to relieve the problem of imbalance between normal and abnormal instances while improving the performance of anomaly detection.
- An LSTM-based detector promotes the efficiency of responding anomalies and marks anomalies of logs instead of detecting session-level anomalies.
- Extensive experiments are conducted to evaluate the effectiveness of LogGAN based on two real-world datasets.

2 Related Work

Generally, the techniques of anomaly detection (i.e., outlier detection) are categorized as supervised, semi-supervised, and unsupervised anomaly detections.

In this section, we will briefly introduce some popular anomaly detections in each category of techniques.

2.1 Supervised Anomaly Detection

Supervised anomaly detections operate under two general assumptions: (1) the labels of normal and abnormal instances are available; (2) the normal and abnormal instances are distinguishable given the feature space. Chen et al. proposed a decision tree-based approach to detecting the actual failures from large Internet sites (i.e., eBay) based on the temporal request traces [5]. The decision trees simultaneously handle the varying types of runtime properties (i.e., continuous and discrete variables). Therefore, the proposed approach was widely used in many practical scenarios. Bodik et al. proposed a fingerprint (i.e., vector) to effectively demonstrate the performance state of systems and implemented a regularized logistic regression-based method for selecting the relevant metrics to build the appropriate fingerprints [1]. The anomalies can be precisely identified using the fingerprints which summarize the properties of the whole data center (e.g., CPU utilization). Liang et al. employed several classifiers (e.g., SVM and nearest neighbor) to detecting the failures in the massive event logs which were collected from the supercomputer IBM BlueGene/L [10]. Similar to Bodik et al., they also derived the specific combination of features to effectively describe each event log for improving the performance of classification tasks, which demonstrates that the representation of normal and abnormal logs is significant. The supervised methods have a quick test phase for the online detections, however, the extreme dependency on the quality of labels limits the application scenarios [18].

2.2 Semi-supervised Anomaly Detection

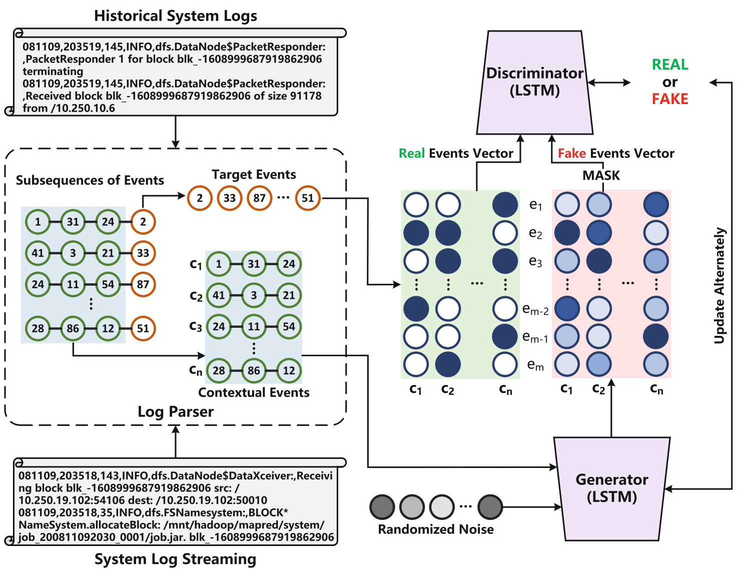
The semi-supervised anomaly detection operates under the assumption: given the feature space, the normal samples are located closely while the anomalies are far from the clusters of normal ones [3]. The representative of the semi-supervised model is the nearest neighbor-based techniques which can be categorized as (1) distance-based neighbors, and (2) density-based neighbors. To address the problem of the high-dimensional feature space, Zhang et al. proposed a High-Dimension Outlying subspace Detection (HighDOD) to searching for the optimal subset of features to represent outliers [20]. Due to the subset of features (i.e., low-dimensional data), the Euclidean distance is capable of describing the actual distance between normal and abnormal instances. Besides distance-based approaches, the density-based method is also useful to distinguish anomalies. To improve Local Outlier Factor (i.e., a type of popular measure to calculating the density given the instance), Chawla et al. proposed a new measure called Spatial Local Outlier Measure (SLOM) [4,14]. Du et al. proposed LSTM-based anomaly detection and diagnosis framework named DeepLog based on unstructured system logs [6]. DeepLog analyzes and detects anomalies using the log key and the parameter value vector to help administrators for diagnosing the system errors based on workflows. DeepLog is trained based on the normal patterns in system logs and provides a way to be incrementally updated using upcoming logs; therefore, DeepLog is categorized as semi-supervised anomaly detection. Tuor et al. also proposed a recurrent neural network-based approach to detecting abnormal instances where the proposed model considered system logs as sentences in language models [16]. Compared to the supervised anomaly detections, semi-supervised techniques do not extremely rely on the labeled data and the distribution of observed instances and outperform the unsupervised approaches generally. However, the selection of measuring distance is significant for the performance of semi-supervised anomaly detections.

2.3 Unsupervised Anomaly Detection

The unsupervised technique is the most popular approach in the domain of anomaly detection because this technique still works even if the label of data is unknown. This characteristic of the unsupervised technique satisfies the assumption that anomalies are generally rare and unknown in practical scenarios. Lin et al. proposed a cluster-based approach (i.e., LogCluster) to addressing the log-based anomalies detection problem based on the data from Microsoft service product teams [11]. LogCluster aims to cluster the historical and upcoming logs using the knowledge base, and engineers only need to distinguish several logs (i.e., events) in each cluster that can identify the type of anomalies which is located in the same cluster. Therefore, it is not necessary to obtain the label of logs, and the similarity between logs is more essential to operate LogCluster. Lou et al. proposed a novel anomaly detection approach to identifying program invariants based on the unstructured console logs [13]. The proposed approach concentrates on structuring the free form description in console logs and mining the meaningful anomalies after grouping the structured logs with parameters. Different from the traditional anomaly detections which construct models fitting normal instances and distinguish instances that do not conform to the constructed model, Liu et al. proposed a novel concept that explicitly isolates abnormal instances [12]. The proposed isolation forest (iForest) is capable of addressing the high-dimensional problems using an attribute selector (i.e., the characteristic of the decision tree). In addition, iForest achieves good performance even if there are no anomalies occurred in the training set. Xu et al. proposed a PCA-based anomaly detection and visualized the promising results using a decision tree [19]. The main contribution of this work is that the source code is considered as a reference to parse console logs for improving the quality of structured data and the quality data will improve the representation of console logs (i.e., extracted distinguishable features). The advantage of unsupervised techniques is that the approaches are independent with the label information of the training set. The disadvantage of unsupervised techniques is that expert knowledge is still needed to utilize unsupervised approaches for detecting anomalies in practical scenarios, although the techniques reduce the massive workloads.

3 Method

In this paper, we propose a generative adversarial network-based anomaly detection approach named LogGAN which improves the performance of identifying anomalies in an adversarial setting. Figure 2 illustrates the overview of LogGAN. The main modules of LogGAN are categorized into three parts:



```
081109,203519,145,INFO,dfs.DataNode$PacketResponder:
,PacketResponder 1 for block blk_-1608999687919862906
terminating
081109,203519,145,INFO,dfs.DataNode$PacketResponder:
,Received block blk_-1608999687919862906 of size 91178
from /10.250.10.6
```

```
081109,203518,143,INFO,dfs.DataNode$DataXceiver:,Receivi
ng block blk_-1608999687919862906 src: /
10.250.19.102:54106 dest: /10.250.19.102:50010
081109,203518,35,INFO,dfs.FSNamesystem:,BLOCK*
NameSystem.allocateBlock: /mnt/hadoop/mapred/system/
job_200811092030_0001/job.jar. blk_-1608999687919862906
```

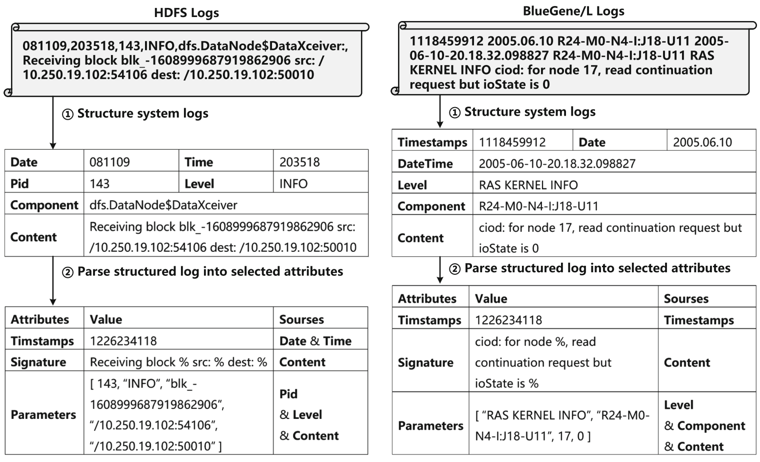
Fig. 2. The framework of anomaly detection generative adversarial network

- *Log Parser:* is the module to parsing unstructured logs into structured logs (or events) which are considered as the minimum units for the following machine learning-based techniques.
- *Adversarial Learning:* is the module to training the LSTM-based anomaly detection model based on the timestamps, signatures, and attributes extracted from structured log.
- *Anomaly Detection:* is the module to detecting and diagnosing anomalies using the LSTM-based model and incrementally update the model based on the upcoming logs and users' feedbacks.

In the following parts of this section, we will introduce each part of LogGAN in detail.

3.1 Log Parser

In the module of log parser, the original unstructured logs are converted into the structured logs. The log parsing, which is considered as the common preprocessing of unstructured logs, is the significant part in the majority of log analysis tasks. Many approaches were proposed to generate events, which are extracted and summarized based on raw logs, for automated performance analysis of system [8,15]. These template-free methods are capable of parsing logs using statistical approaches. However, the performance of these methods is not convincing, because the formations of logs from different systems are chaotic and that is nontrivial to be captured. Therefore, in this paper, we first divide the unstructured logs into several parts (e.g., datetime and content) using the corresponding template, then further extract meaningful information (i.e., event) from these parts [21]. Generally, the event consists of three major components: (1) timestamps, (2) signature and (3) parameters. To make readers fully understand the process of log parser, Fig. 3 illustrates the examples of parsing unstructured logs from two real-world systems (i.e., HDFS and BlusGene/L), respectively.


HDFS Logs
① Structure system logs
② Parse structured log into selected attributes
BlueGene/L Logs
① Structure system logs
② Parse structured log into selected attributes

```
081109,203518,143,INFO,dfs.DataNode$DataXceiver:,
Receiving block blk_-1608999687919862906 src: /
10.250.19.102:54106 dest: /10.250.19.102:50010
```

Date	081109	**Time**	203518
Pid	143	**Level**	INFO
Component	dfs.DataNode$DataXceiver		
Content	Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010		

Attributes	**Value**	**Sourses**
Timstamps	1226234118	**Date** & **Time**
Signature	Receiving block % src: % dest: %	**Content**
Parameters	[143, "INFO", "blk_-1608999687919862906", "/10.250.19.102:54106", "/10.250.19.102:50010"]	**Pid** & **Level** & **Content**

(a) HDFS

```
1118459912 2005.06.10 R24-M0-N4-I:J18-U11 2005-
06-10-20.18.32.098827 R24-M0-N4-I:J18-U11 RAS
KERNEL INFO ciod: for node 17, read continuation
request but ioState is 0
```

Timestamps	1118459912	**Date**	2005.06.10
DateTime	2005-06-10-20.18.32.098827		
Level	RAS KERNEL INFO		
Component	R24-M0-N4-I:J18-U11		
Content	ciod: for node 17, read continuation request but ioState is 0		

Attributes	**Value**	**Sourses**
Timstamps	1226234118	**Timestamps**
Signature	ciod: for node %, read continuation request but ioState is %	**Content**
Parameters	["RAS KERNEL INFO", "R24-M0-N4-I:J18-U11", 17, 0]	**Level** & **Component** & **Content**

(b) BlusGene/L

Fig. 3. Example of log parser to converting from logs to structured entities

Note that, HDFS and BlusGene/L are different in the system structures and workflows, hence the parsed structures from the first step are also different. Observed from Fig. 3a, the timestamps, signature, and parameters are extracted exactly where the signature is a static content that presents a type of logs and the parameters record dynamic parts in each log. The three-tuple representation (i.e., timestamps, signature, and parameters) effectively describes the status of each event which provides administrators with sufficient references to diagnose the broken-down system.

3.2 Adversarial Learning

In this paper, we cast the task of anomaly detection as a set of adversarial learning and propose an LSTM-base generative adversarial network named LogGAN to improve the performance of identifying anomalies. The concept of the generative adversarial network (GAN) was proposed by Goodfellow et al. where GAN considers a machine learning problem as a game between two models (i.e., generator and discriminator) [7]. The generator (G) captures the distribution of real samples and generates plausible samples which are similar with real samples in the representation of features, while the discriminator (D) tries to identify whether the upcoming sample is real or synthetic one for improving the quality of samples generated by G. The iteration repeats until both G and D converge, then G is capable of generating 'real' samples. This game setting of machine learning exactly addresses a significant problem in anomaly detection: the overwhelming ratio of normal and abnormal instances. The fully-trained G can capture the distribution of anomalies which further improves the performance of detecting whether the upcoming log is normal or abnormal.

The original GAN, which is utilized to generate continuous variables of images, do not match the scenario of predicting discrete event ID (i.e., signature) [7]. Therefore, we propose LogGAN to independently generate the continuous probability of each upcoming event instead of using the softmax layer to output the probability distribution of overall events [2,17]. In details, given an observed set of temporal events $\mathbf{S} = \{e_{(1)}, e_{(2)}, ..., e_{(s)}\}$ from parsed system logs and a set of event $\mathbf{E} = \{e_1, e_2, ..., e_m\}$ where e_j presents a signature of the j_{th} event, the task of LogGAN is to predict whether the upcoming event (i.e., log) is normal or abnormal based on the context combinations from the set $\mathbf{C} = \{c_1, c_2, ..., c_n\}$ where c_i demonstrates the i_{th} combination $(e_{(k-2)}, e_{(k-1)}, e_{(k)})$ within a 3-size sliding window. As a game setting, we exploit Long Short Term Memory network (LSTM) for both G and D where G aims to generate fake normal and abnormal instances and D tries to distinguish whether the instance is real or fake. For G, we utilize a random noise $\mathbf{z}$ and a combination c_i as the input of LSTM[1] while the output is an $m-$dimensional vector representing the independent occurring probability of each event in $\mathbf{E}$. For D, we utilize a combination c_i as the input and an $m-$dimensional vector of the independent occurring probability as the parameter[2] of LSTM while the output is whether the $m-$dimensional vector is real or fake sample under the contextual combination c_i. Therefore, the objective function of G and D is defined as follows, respectively:

$$\begin{aligned} J^G &= \min_\theta \sum_{i=1}^{n} (\mathbb{E}_{\hat{\mathbf{e}} \sim P_\theta}[\log(1 - D(\hat{\mathbf{e}}|\mathbf{c}))] + \sum_{j=1}^{m} (\hat{e_j} - e_j)^2) \\ &= \min_\theta \sum_{i=1}^{n} (\log(1 - D(\hat{\mathbf{e_{c_i}}}|\mathbf{c_i})) + \frac{1}{m} \sum_{j=1}^{m} (\hat{e_{c_i j}} - e_{c_i j})^2)), \end{aligned} \tag{1}$$

[1] Learned event embedding is used to demonstrate each event.

[2] In D, we cast the combination c_i as the input of LSTM and LSTM directly outputs the hidden layer without any manipulation. Then, we concatenate the $m-$dimensional vector with the hidden layer as an input of a two-layer full Connected neural network which outputs whether the $m-$dimensional vector is real or fake as a binary classification.

$$\begin{aligned} J^D &= \min_{\phi} - \sum_{i=1}^{n} (\mathbb{E}_{\mathbf{e} \sim P_{true}}[\log D(\mathbf{e}|\mathbf{c})] + \mathbb{E}_{\hat{\mathbf{e}} \sim P_{\theta}}[\log(1 - D(\hat{\mathbf{e}}|\mathbf{c}))]) \\ &= \min_{\phi} - \sum_{i=1}^{n} (\log D(\mathbf{e_{c_i}}|\mathbf{c_i}) + \log(1 - D(\hat{\mathbf{e_{c_i}}}|\mathbf{c_i}))), \end{aligned} \tag{2}$$

where θ and ϕ is the parameter of G and D, respectively. Note that, $\hat{\mathbf{e_{c_i}}} = \mathbf{e}'_{\mathbf{c_i}} \odot \mathbf{o_{c_i}}$ is an $m-$dimensional vector representing the independent occurring probability of each event in $\mathbf{E}$ (i.e., input of D), where $\mathbf{e}'_{\mathbf{c_i}}$ is the output of G and $\odot$ is the element-wise mask multiplication. $\mathbf{o_{c_i}}$, which is an $m-$dimensional observed vector (i.e., o_{c_ij} stands for the observation of e_j where $o_j \in \{1, 0\}$ represents whether e_j is an upcoming event next to $\mathbf{c_i}$ or not), is used to filter the occurring probability of unobserved events in $\mathbf{e}'_{\mathbf{c_i}}$. This setting assists LogGAN to only update the gradients based on the loss of observed events (i.e., both normal and abnormal instances) and avoid the disturbance generated by the unobserved one. In addition, during the process of updating G, we apply a reconstruction error (i.e., $\sum_{j=1}^{m}(\hat{e_{c_ij}} - e_{c_ij})^2$) to help G capture the actual distribution of training data for further improving the performance. Algorithm 1 shows the overall algorithm of LogGAN in detail.

Algorithm 1. The algorithm of LogGAN

```
Input:
    $G_\theta$: the generator $G$,
    $D_\phi$: the discriminator $D$,
    $B$: the size of minibatch,
    $N$: the number of maximum iteration.
Output:
    $G_{\theta^*}$: converged generator $G$.
 1: Initialize $G_\theta$ and $D_\phi$ with random weights $\theta$ and $\phi$.
 2: Set $t \leftarrow 0$
 3: repeat
 4:     for G-steps do
 5:         Sample $B$ combinations of events as a minibatch $M_G$
 6:         Generate corresponding fake instances using generator $G_\theta$ and train $G_\theta$
 7:         Update $G_\theta$ by $\theta^* \leftarrow \theta - \frac{1}{B}\nabla_\theta J^G$
 8:     end for
 9:     for D-steps do
10:         Sample $B$ combinations of events as a minibatch $M_D$
11:         Generate corresponding fake instances using generator $G_\theta$
12:         Combine the generated instances with sampled real instances and train $D_\phi$
13:         Update $D_\phi$ by $\phi^* \leftarrow \phi - \frac{1}{B}\nabla_\phi J^D$
14:     end for
15:     Update $t \leftarrow t + 1$
16: until LogGAN converges OR $t >= N$
17: return $G_{\theta^*}$.
```

Negative Sampling: In practical scenarios, given a combination of events, the possible upcoming events are sparse. In other words, the real event vector (i.e., $\mathbf{e_{c_i}}$) is more like a one-hot or multi-hot encoding vector which causes the over-fitting problem. Therefore, we exploit a negative sampling strategy to avoid the overfitting problem [2]. During the G-steps, we randomly sample the unobserved instances according to a specific ratio and set the corresponding position of mask o_{c_i} as 1 for retaining the gradients.

3.3 Anomaly Detection

After completing the training LogGAN, generator G is applied to detect anomalies based on the streaming events from system logs. During the stage of anomaly detection: (1) the historical and upcoming system logs are transformed into structured data (i.e., event) via the log parser; (2) the input of G is the combination of several latest events (i.e., several one-hot encoding vectors) and G generates a corresponding $m-$dimensional vector representing the independent occurring probability of each event; (3) a set of normal events is built based on the generated $m-$dimensional vector filtered using a predefined threshold of normal probability in the step 2; (4) the upcoming event is considered as a normal instance if the event has an intersection with the set of normal events, otherwise, the event will be alerted as an anomaly.

4 Experiment

In this section, we propose the experiments to evaluate the effectiveness of LogGAN on two real-world datasets, and mainly concentrate on the following issues:

- *Parameter:* We analyze the effect of different parameters on the performance of LogGAN.
- *Session-level Anomaly Detection:* The performance of LogGAN on the task of session-level anomaly detection is compared to that of baselines.
- *Log-level Anomaly Detection:* The performance of LogGAN on the task of log-level anomaly detection is compared to the performance of DeepLog.

4.1 Experimental Setup

Datasets: Generally, up-to-date system logs are rarely published and are sensitive data that describe the detailed information (i.e., business and transaction) about the deployed large scale system, however, the data collected from own small scale system hardly show the actual anomalies in practical scenarios. Therefore, we exploit two real-world datasets (i.e., HDFD and BGL) collected several years ago which is published for research [21]. HDFS is collected from Amazon EC2 platform where 11,197,705 system logs are divided into 575,139 sessions and generated by 203 nodes during two days while BGL contains 4,747,963 logs collected from the BlueGene/L supercomputer system during 215 days. The detailed information of datasets is shown in Table 1.

Baselines: In the experiments, to evaluate the performance of our proposed approach, we compare LogGAN with several selected baselines:

Table 1. The overview of two real-world datasets

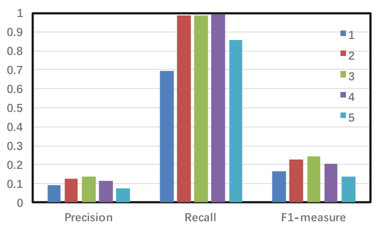
System	Start date	Days	Size (GB)	Rate (log/sec)	Messages	Alerts	Signatures
HDFS	2008-11-09	2	1.490G	64.802	11,197,705	16,916/575,139	29
BGL	2005-06-03	215	0.708G	0.256	4,747,963	348,698	394

- iForest [12]: is an *unsupervised* tree-based isolation forest which tries to isolate anomalies from other normal instances, especially for the imbalanced training set.
- PCA [19]: is an *unsupervised* principal component analysis-based anomaly detection technique which improves the parser of unstructured systems and visualizes the promising diagnosis of abnormal instances.
- Invariants Mining [13]: is an *unsupervised* anomaly detection technique applied to build the structured logs based on the unstructured description in console logs.
- LogCluster [11]: is an *unsupervised* cluster-based approach to clustering events extracted from the historical and upcoming logs based on the knowledge base.
- DeepLog [6]: is a *supervised* LSTM-based deep learning framework which utilizes LSTM to fit the distribution of normal instances using the log key and the performance value vector extracted from each log.

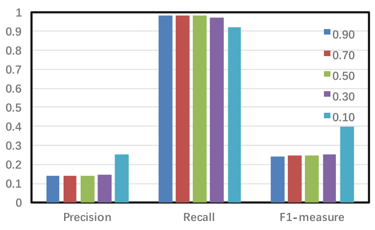
In the experiments, we exploit the first 30% of dataset as the training set while the remaining data as the test set based on time series. In addition, we will briefly introduce the key parameters of LogGAN for the reproduction of our model. The size of sliding window determines the capacity of contextual events to the upcoming log. The larger size demonstrates the more specific contextual patterns are used to identify anomalies while the smaller size means upcoming anomalies are determined by the latest events (i.e., the more regular contextual patterns). The event embedding is used to represent events in the continuous space. In this paper, we utilize the 3-size sliding window to extracting contextual pattern of upcoming logs. To distinguish normal and abnormal events from the output of generator (i.e., an $m-$dimensional vector), we define a threshold to filtering normal logs. When the occurring probability of a event is below the predefined threshold, the event is considered as an anomaly based on the contextual pattern. In addition, we define the threshold as 0.90 which means the upcoming log is normal if the appearing probability of the log is 90% based on the output of generator. The ratio of negative sampling is set as 0.1. The 2-layer LSTM is applied as the basic model of generator and discriminator in LogGAN. The dimension of event embedding is set as 200. To keep the correspondence with DeepLog, in the experiments, we define the accurate identification of true anomalies as the true positive. Therefore, the metrics (e.g., precision and recall) demonstrate the performance of detecting anomalies.

4.2 Result and Discussion

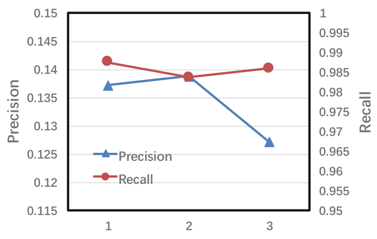
Parameters: Figure 4 illustrates the performance of LogGAN within different settings of parameters including the size of sliding window, the threshold to filtering normal logs, and the layer of LSTM. In this section, we concentrate on the task of log-level anomaly detection. First, Fig. 4a shows the performance (i.e., Precision, Recall, and F1-measure) of LogGAN on different sizes of sliding window (i.e., size 1 to 5). Observed from Fig. 4a, abnormal logs are correlated with the appropriate context of events (i.e., 3-size sliding window). Neither the concurrence of pair-wise events (i.e., 1-size sliding window) nor the extremely specific contextual pattern (i.e., 5-size sliding window) is beneficial to identity log-level anomalies. Second, Fig. 4b illustrates the performance on different settings of threshold to filtering normal logs. Note that, LogGAN has the similar performance on the threshold from 0.90 to 0.30 while the performance becomes worse when the threshold is 0.10. In other word, the appearing probability of normal and abnormal logs largely depends on whether the combination of contextual events and logs occurs in the training set. Finally, Fig. 4c shows the performance of LogGAN using different layers of LSTM in the generator and discriminator. The experimental results demonstrate that appropriately using deep features (i.e., 2-layer LSTM) is capable of improving the performance of detecting anomalies.



(a) Size of sliding window



(b) Threshold to filtering normal logs



(c) Layer of LSTM

Fig. 4. The performance of LogGAN within different settings of parameters

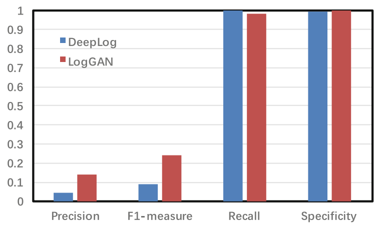


Fig. 5. The comparison between DeepLog and LogGAN on log-level anomaly detection

Log-level Anomaly Detection: Figure 5 illustrates the comparison between DeepLog and LogGAN on the log-level anomaly detection on BGL. The characteristic of DeepLog is to utilize feedbacks (i.e., false positive samples) from administrators to update the model incrementally. In other words, DeepLog aims to learn the whole normal instances including the upcoming ones to detect anomalies without considering the correlation between normal and abnormal logs. Any method, which has the strategy of incremental learning, has the benefit of this setting including LogGAN. In this experiment, we concern more about the generalization ability of model based on the limited training set. Observed from Fig. 5, LogGAN outperforms DeepLog on the task of anomaly detection based on the same size of training set (i.e., 30%). However, the overall performance of DeepLog and LogGAN is not satisfactory. To further improve the performance of anomaly detection, we need to extract more meaningful feature from logs instead of only using the sequential information.

Table 2. The comparison between baselines and LogGAN on session-level anomaly detection on HDFS

Method	Recall	Precision	F1-score
Invariants miner	**1.000**	0.084	0.154
PCA	0.346	0.707	0.465
DeepLog	0.016	0.939	0.032
iForest	0.318	**1.000**	0.482
LogClustering	0.362	**1.000**	**0.532**
LogGAN-sess	0.356	**1.000**	**0.525**

Session-level Anomaly Detection: Table 2 shows the performance of baselines and LogGAN on HDFS dataset. Different from the version of LogGAN used in the log-level anomaly detection, we propose a session-level version of LogGAN (LogGAN-sess) in the session-level task. The generator of LogGAN-sess aims to match a $30-$dimensional vector where the first 29 dimensions record the number of corresponding events appeared in the current session, and the last one represents the abnormal score instead of fitting an $m-$dimensional vector. The experimental results show that LogGAN-sess outperform other baselines except for LogClustering. The limitation of current LogGAN-sess is the model only exploits the statistics of independent event that occurred in the session. However, the traditional anomaly detection methods concentrate on the concurrence of several events in the temporal sequence. In addition, the structure of workflows is also significant information which describes the normal and integrated transactions in the system. Therefore, the performance of LogGAN-sess could be further improved using the statistics of specific patterns (i.e., the combination of temporal logs).

5 Conclusion

To overcome the limitation of diagnosing log-level anomaly detection, in this paper, we propose a sequence-based generative adversarial network to detecting abnormal events among system logs named LogGAN. In practical scenarios, we consider that the occurring anomalies depend on specific patterns which comprise the latest logs and regard specific patterns as the contextual information of upcoming logs. Due to the benefit of the generative adversarial network, the problem of the imbalance between normal and abnormal logs is relieved where LogGAN is capable of generating 'real' anomalies for supplying the lack of abnormal logs in system logs. In addition, LogGAN can be transformed into the session version only to changing the representation of samples without reforming the overall structure of LogGAN. The experimental results show the effectiveness of LogGAN on both the tasks of session-level and log-level anomaly detection.

The current LogGAN still has some problems that need to be solved for further improvement, and there exist several ideas to extend our work in the future. First, the current LogGAN has similar structures of discriminator and generator, and we exploit the generator to distinguish anomalies from system logs. Can the combination of outputs from discriminator and generator be used to identify anomalies? Second, only the signature and the temporal information of system logs are used to train LogGAN in this paper. The parameter of each event and other meaningful feature need to be considered to precisely describe anomalies. Third, the diagnosis of anomalies is also an important task which helps administrators solve anomalies efficiently. Therefore, the root cause analysis (RCA) should be considered in the process of detecting anomalies.

Acknowledgment. This work was supported by the National Natural Science Foundation of China under Grant No. 61802205, 61872186, and 61772284, the Natural Science Research Project of Jiangsu Province under Grant 18KJB520037, and the research funds of NJUPT under Grant NY218116.

References

1. 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. ACM (2010)

2. Chae, D.K., Kang, J.S., Kim, S.W., Lee, J.T.: CFGAN: a generic collaborative filtering framework based on generative adversarial networks. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, pp. 137–146. ACM (2018)
3. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: a survey. ACM Comput. Surv. (CSUR) **41**(3), 15 (2009)
4. Chawla, S., Sun, P.: SLOM: a new measure for local spatial outliers. Knowl. Inf. Syst. **9**(4), 412–429 (2006)
5. Chen, M., Zheng, A.X., Lloyd, J., Jordan, M.I., Brewer, E.: Failure diagnosis using decision trees. In: International Conference on Autonomic Computing. Proceedings, pp. 36–43. IEEE (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. ACM (2017)
7. Goodfellow, I.J., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, Montreal, Quebec, Canada, 8–13 December 2014, pp. 2672–2680 (2014). http://papers.nips.cc/paper/5423-generative-adversarial-nets
8. Guo, S., Liu, Z., Chen, W., Li, T.: Event extraction from streaming system logs. In: Information Science and Applications 2018 - ICISA 2018, Hong Kong, China, 25–27th June 2018, pp. 465–474 (2018). https://doi.org/10.1007/978-981-13-1056-0_47
9. Li, T., et al.: FIU-Miner (a fast, integrated, and user-friendly system for data mining) and its applications. Knowl. Inf. Syst. **52**(2), 411–443 (2017)
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. Lin, Q., Zhang, H., Lou, J.G., Zhang, Y., Chen, X.: Log clustering based problem identification for online service systems. In: Proceedings of the 38th International Conference on Software Engineering Companion, pp. 102–111. ACM (2016)
12. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining, pp. 413–422. IEEE (2008)
13. 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)
14. Sun, P., Chawla, S.: On local spatial outliers. In: Fourth IEEE International Conference on Data Mining (ICDM 2004), pp. 209–216. IEEE (2004)
15. 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, pp. 785–794. ACM (2011)
16. Tuor, A.R., Baerwolf, R., Knowles, N., Hutchinson, B., Nichols, N., Jasper, R.: Recurrent neural network language models for open vocabulary event-level cyber anomaly detection. In: Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence (2018)
17. Wang, J., et al.: IRGAN: a minimax game for unifying generative and discriminative information retrieval models. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 515–524. ACM (2017)

18. Xia, B., Li, T., Zhou, Q.F., Li, Q., Zhang, H.: An effective classification-based framework for predicting cloud capacity demand in cloud services. IEEE Trans. Serv. Comput. (2018)
19. 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. ACM (2009)
20. Zhang, J., Wang, H.: Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance. Knowl. Inf. Syst. **10**(3), 333–355 (2006)
21. Zhu, J., et al.: Tools and benchmarks for automated log parsing. CoRR abs/1811.03509 (2018). http://arxiv.org/abs/1811.03509