2018 IEEE International Conference on Systems, Man, and Cybernetics

A Convolutional Auto-encoder Method for Anomaly Detection on System Logs

Yu Cui*†, Yiping Sun*†, Jinglu Hu* and Gehao Sheng†
*Graduate School of Information, Product and Systems, Waseda University,
2-7 Hibikino, Wakamatsu, Kitakyushu-shi, Fukuoka, JAPAN
†School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University,
800 Dongchuan Road, Minhang District, Shanghai, CHINA
Email: adacui523@toki.waseda.jp, sunacc@suou.waseda.jp, jinglu@waseda.jp, shenghe@sjtu.edu.cn

***Abstract*—Anomaly detection on system logs is to report system failures with utilization of console logs collected from devices, which ensures the reliability of systems. Most previous researches split logs into sequential time windows and regarded each window as an independent instance for classification using popular machine learning methods like support vector machine(SVM), however, neglected the time patterns under logs. Those approaches also suffer from information loss due to the vector representation, and high dimensionality if there is a large number of log events. To make up these deficiencies, unlike most traditional methods that used a vector to represent a period behavior at the macro level, we construct a 2D matrix to reveal more detailed system behaviors in the time period by dividing each window into sequential subwindows. To provide a more efficient representation, we further use the ant colony optimization algorithm to find a highly-coupled event template as the horizontal index of the 2D window matrix to replace the disordered one. To capture time dependencies, a multi-module convolutional auto-encoder is configured as that different paralleled modules scan among different time intervals to extract information respectively. These features are then concatenated in latent space as the final input, which contains diversified time information, for classification by SVM. The experiments on Blue Gene/L log dataset showed that our proposed method outperforms the state-of-art SVM method.**

***Index Terms*—Log Analysis; Anomaly Detection; Feature Extraction; Auto-encoder; Ant Colony Optimization**

I. Introduction

It is now the era of information that an observable quantity of manual work has been replaced by the effort of information technology. From the most basic programmable logic control to the marvelous inventions like artificial intelligence, computing systems have achieved a monopoly position, attributed to low cost and high speed. The reliability of system is of vital importance that once systems went down, it would cause severe impact on economics and security. For example, in 2010, a flash crash led to 1,000 points drop in the New York Stock Exchange. The computer network of United Airlines broke down in 2012, bringing a one-month flight delays [1].

One way to improve reliability of computing systems is reducing the downtime of systems by intermediate anomaly detection which provides crucial aid for inspection and repair. Anomaly detection [2] is referred to the clarification of items, events or observations which do not conform to an expected pattern or other items in a dataset. The most popular media tool to detect system failures is console logs which are generated through the whole working time of systems, including runtime information of operation status, time point, task event and execution details. By summarizing the similar characteristics of historical anomalous log sequences, failure could be such defined for classifying the newly generated log flows.

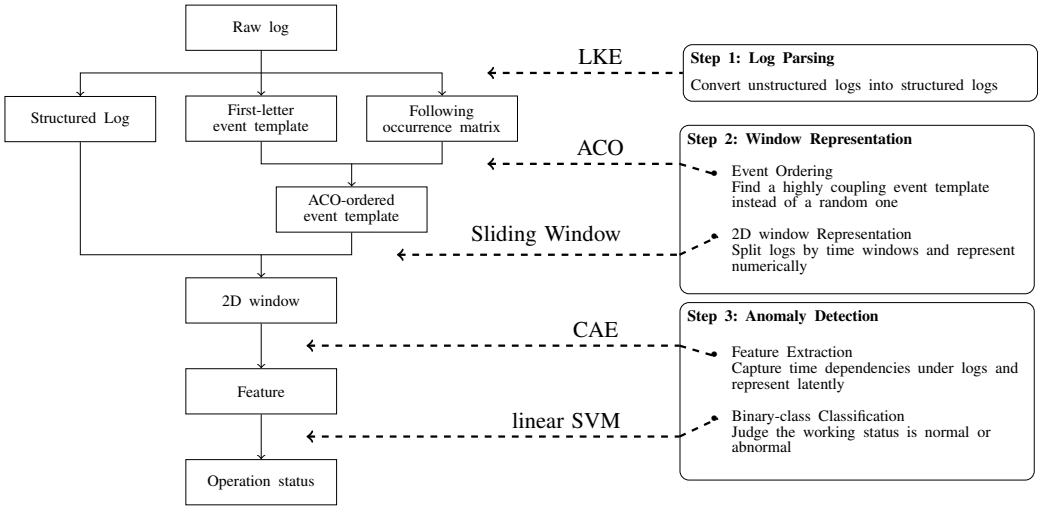
In the earliest time, the failure detection was accomplished by manual inspection, where simple code was used to find specific words like error and fatal. With the widespread of large-scale modern systems, it has been eliminated due to the explosion of log lines. In addition to the development of tolerance mechanism, logs with those error-meaning words might not cause accident to the system. The simple word match method would report those tolerated points also as outliers, decreasing the precision of anomaly detection [3].

Hereafter, researches mainly relied on threshold and statistical analysis, where text logs were represented by event numbers. Cumulative sum and weighted moving average over time intervals were used for outlier evaluation by comparing to a predefined threshold [4]. This method is influenced greatly by the predefined threshold and the complexity of log patterns.

With the bloom in artificial intelligence, some machine learning methods become solutions to this log failure detection problem, such as support vector machine and decision tree. Firstly, text logs sequences are parsed into structured logs, and divided by time windows. Then each window is expressed by a numerical vector and fed into the machine learning classifiers [5]. However, log sequences are treated as independent instances, ignoring the former log history. When the window size becomes larger, the information loss on time-series is more severe. Besides, when there are hundreds of log event types, the performance would also drop to some extent.

To make full use of time information and deal with the diversified category of log events, we propose a 2D window matrix form instead of the accustomed vector. As the order of columns in 2D window has a profound impact on the effectiveness of representation(e.g. imagine a picture whose columns of pixels are randomly shuffled), we borrow the concept of ant colony optimization(ACO) method to find a highly-coupled log event series instead of a random ordered one. A multi-module convolutional auto-encoder(CAE) is utilized to extract time features and compress the input to a low-dimensional

2577-1655/18/$31.00 ©2018 IEEE
DOI 10.1109/SMC.2018.00519



(a) Data flow (b) step flow

Fig. 1. Framework of Proposed Method for Anomaly Detection on System Logs

latent expression. Moreover, the application of parallel multi-module assists to capture various information through different time intervals. The extracted feature is then fed into the support vector machine(SVM) to classify the operation status.

The rest of paper is organized as follows. In Section II, the framework of proposed method would be briefly introduced. The details of multi-module convolutional auto-encoder feature extraction model is demonstrated in Section III. Section IV shows the configuration of our model and analyzes the experiment result. Conclusions are drawn out in section V.

II. FRAMEWORK

The brief procedure of our proposed method for system log-based anomaly detection is depicted in Fig 1, consisting of 3 steps: log parsing, window representation and anomaly detection. In our proposed method, we remain the first step as traditional methods, and come up with a 2D window matrix representation form rather than vectors to reduce time information loss. To further extract the potential time patterns under 2D windows, we make improvement from 2 aspects, providing a more efficient representation and capturing time dependencies among different time intervals.

A. Log Parsing

Although console logs store runtime information at each time point, they could not be directly used for classification due to their free text form. As shown in Fig 2, raw logs contain a lot of redundant information like port numbers and hexadecimal values, which are dynamically generated to demonstrate working details. Log parsing is to remove these unimportant information and leave the essence of executions called log events of tasks. In our work, we choose Log Key Extraction (LKE) [6] for log parsing. The relative distance matrix of all logs are calculated to determine similarities between log message pairs. Then K-means clustering algorithm [7] is used for splitting. In this step, we could acquire structured logs and an event template ordered by first letter.

B. Window Representation

As large-scale system generates tons of logs within seconds [8], and the interval of neighboring logs varies from operating conditions, it is not that reasonable to treat each log as an instance. On demand of detecting frequency [9], windowing method is applied that logs within a predefined time period T are gathered as a window sample. In each time window, the times each event type has occurred is respectively recorded in an event count vector [10]. While these vector instances are fed into classifiers like SVM, decision tree, they are actually regarded as the independent samples, without considering time patterns. To address this problem, we propose a 2-dimensional window matrix that separates the large time interval T into smaller equal-length subwindows instead. In the 2D matrix representation, the horizontal index matters indeed since events are not independent variables but related to each other. Thus, we introduced the ant colony optimization algorithm to find a highly-coupled sequence of event template to replace the one ordered by first letter.

1) 2-dimensional Window Representation:

The core idea of 2D window matrix is that the macro behavior in a period could be constituted by micro behaviors in multiple continuous time infinitesimals. For given log dataset X, time window with span of T splits whole dataset as $X = (x_1, x_2, ..., x_i, ...)$, each instance $x_i \in X$ is denoted as a numerical representation of a time window. Traditionally, each time window was represented by an event count vector, that $x_i \in \mathcal{R}^{1\times M}$, where M is the number of different events.

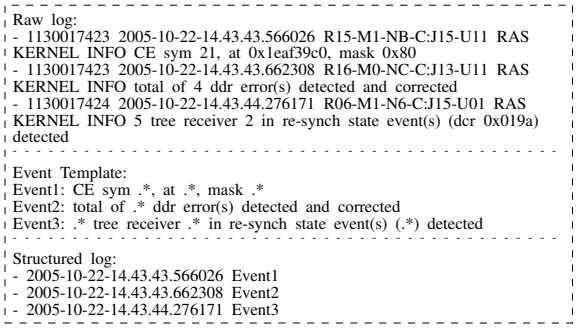
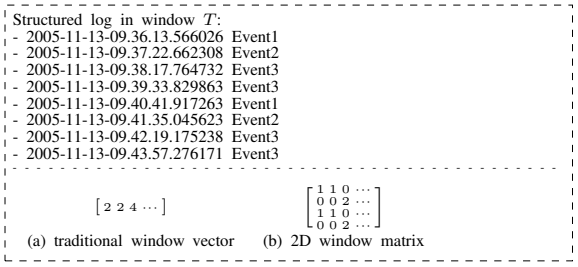


Fig. 2. Demonstration of Logs in Text Form and Event Template



(a) traditional window vector (b) 2D window matrix

Fig. 3. Information Loss in Traditional Vector Representation

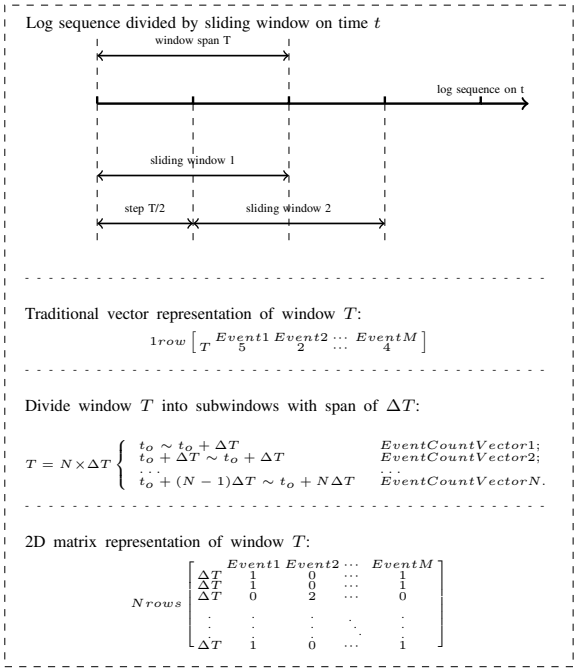


Fig. 4. Concept of 2D Window Representation

As shown in Fig 4, in 2D window, period T is represented by N event count vector in ΔT subwindows, that $x_i \in \mathcal{R}^{N \times M}$.

In Fig 3, the simple event count vector in accustomed method is of a such vague expression that the order of event occurrence is neglected since there is no evidence that event3 commonly occurs after event1 and event2. However, this information could be stretched out if we use 2D window matrix where $\Delta T = 2\,\mathrm{min}, N = 4$. It is indicated that 2D window representation contains much more time information, while conventional one does not so. Commonly larger the N, more detailed time information would be covered. However, noise would be amplified if ΔT is too small since in real cases, one log event would be generated continuously until the task is accomplished that there would only be 1 event type in a series of subwindows. Therefore, N would better be predefined empirically and prudentially.

2) Ant Colony Optimization for Event Ordering:

As execution jobs are logical programming commands, there exist complex relationships between log events. We intend to find an event order that could reveal the closest relationships between events as the horizontal index for our 2D window matrix.

To quantify relationship between events, the following occurrence times O_{ij} is used as the relationship score, where O_{ij} means how many times event E_j occurs after event E_i, $i \neq j$. Inspired by the application of ant colony optimization (ACO) to the traveling salesman problem [11], the highly-coupled event sequence with the maximal relationship score could be discovered through the parallel search of ants.

$$R_S = \max \sum_{k=1}^{M-1} O_{S_k S_{k+1}} \tag{1}$$

With event template set E_M consisting of M events and the following occurrence time matrix $O \in \mathcal{R}^{M \times M}$, denote the heuristic factor is proportional to following occurrence time:

$$\eta_{ij} = \frac{O_{ij}}{\max O_i} \tag{2}$$

Initialize the pheromone reward factor as $\tau_{ij}(0) = 1$ and update with the best sequence of each iteration $S_{iterbest}$ weighted by evaporation rate ρ and global gain Q:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho \frac{Q}{R_{S_{iterbest}}} \tag{3}$$

For tail event E_i in S_{temp}, the probability of next event E_j to be appended after E_i from the rest events set $E_{rest} = E_M - S_{temp}$ is shown as:

$$P_{i,j}(t) = \frac{\tau_{ij}^{\alpha}(t) \times \eta_{ij}^{\beta}}{\sum_{k \in E_{rest}} \tau_{ik}^{\alpha}(t) \times \eta_{ik}^{\beta}} \tag{4}$$

where α is the coefficient on pheromone reward factor, and β is the coefficient on heuristic factor.

On behalf of that $P_{i,j}(t)$ might equal to 0 because $\eta_{ij} = 0$ as E_j never occurs after event E_i, the state transition rule based on random-proportional rule is as follows:

$$E_j = \begin{cases} \arg\max[P_{i,j}(t)], & \eta_{ik}^{\beta} > 0, q \geq q_0 \\ \arg\max[\tau_{ij}^{\alpha}(t) \times \eta_{ij}^{\beta}], & \eta_{ik}^{\beta} > 0, q < q_0 \\ \arg\min[j-i], & \text{otherwise} \end{cases} \tag{5}$$

where q is a random number uniformly distributed in $[0, 1]$, $0 \leq q_0 \leq 1$ is the predefined exploitation parameter.

C. Anomaly Detection

This part could be regarded as a binary-class classification of normal and abnormal operation status. [12] applied RIPPER (a rule-based classifier), support vector machine and nearest neighbor method to operation logs. [13] visualized a general system log anomaly detection model on decision tree. [14] realized anomaly detection on console logs by invariants mining which reveals the inherent linear characteristics of program work flows. [3] eased the log-based problem by an unsupervised learning method which clusters logs into multiple clusters. [5] evaluated several machine learning techniques: decision tree, SVM, log clustering, principal component analysis and so on. However, these above methods missed the time patterns pitifully.

In our model, before classification, a multi-module convolutional auto-encoder is constructed to capture time-dependencies in the 2D window matrix and give a latent feature representation of behaviors with unit of window period. Then the extracted feature is fed into a SVM classier for detection. Experiments have been done to verify our approach by comparing results to traditional method that each window is expressed in a window vector and then directly fed into SVM as an independent sample.

III. Anomaly Detection

A. Convolutional Auto-encoder for Feature Extraction

Auto-encoder [15] is an unsupervised learning structure consisting of an encoder and a decoder. The encoder part first maps the input into lower dimensional representation in latent space. Then the decoder network reconstructs the input by a reverse mapping. As the target output is assigned as the input without feeding with labels, the compressed feature representation is learned by diminishing the reconstruction error through the back propagation.

Unlike fully connected layers in auto-encoder, the convolutional auto-encoder architecture [16] uses the convolutional layers for encoding and deconvolutional layers for decoding. The weight sharing among positions promotes CAE to absorb the localized common patterns at time points of our log messages.

To further absorb information among different time intervals, the multi-module CAE shown in Fig 5 is configured as that the encoder and decoder are both with 3 parallel convolutional modules with different filter sizes.

For an instance input $x_i \in \mathcal{R}^{N \times M}$, the horizontal axis is indexed by the ACO-optimized log event order while the vertical axis is the time sequence in the period. Filters of different module share identical width W on event dimension, but possess unequal height on time dimension. The height of 1 is to find the features of each independent time point, while the one of 2 is to dig out correlations between two adjacent time points. A more macro scope of time patterns of this time period is scaled by half of the window $\frac{T}{2}$. The numerical results

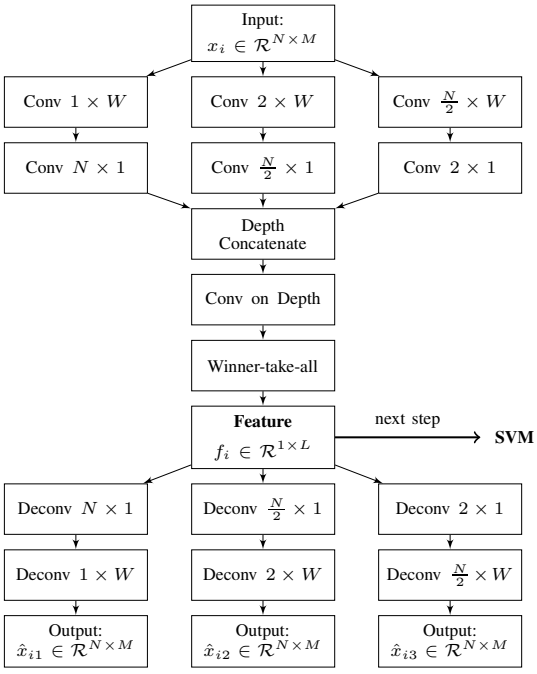


Fig. 5. Construction of Multi-Module Convolutional Auto-encoder

after a module convolutional calculation are designed with same dimension on height, width and depth for concatenation on depth. Then the temporary concatenated feature would be compressed by a convolutional filter on channels into a single channel expression.

The winner-take-all activation [17] is used in the latent feature layer to prevent over-fitting, where only the top $k\%$ largest unit values are kept, leaving the rest to zero. This could be regarded as a sort of regularization, ensuring a constant level of sparsity.

In decoder part, the feature $f_i \in \mathcal{R}^{1 \times L}$ encoded is fed into 3 modules corresponding to those in the encoder to approximate input x_i by decreasing reconstruction error of 3 outputs, x_{i1}, x_{i2} and x_{i3}:

$$J = \sum (x_i - \hat{x}_{i1})^2 + \sum (x_i - \hat{x}_{i2})^2 + \sum (x_i - \hat{x}_{i3})^2 \quad (6)$$

B. Support Vector Machine for Binary-class Classification

Support Vector Machine(SVM) is a popular supervised method in classification field. In this algorithm, one or more hyperplanes would be constructed in a high dimension space. Hyperplanes are determined under the condition that the margin between them and the nearest point of the other categories is of the maximal value [5]. To detect failures, training samples of features exported from CAE and corresponding labels construct the hyperplanes. For a new coming window instance, the relative position of it and the configured hyperplanes is used to determine which category it belongs to.

IV. EXPERIMENTS

A. Experiments Setup

1) Log Dataset:

Blue Gene/L [18] dataset is a publicly available system log dataset and is widely studied for log analysis in [5], [12]. 4,747,963 log messages are recorded through 215 days from 3rd June 2015, with 345,460 log pieces labeled as outliers. Through process, 80% data are used for training, and the remaining 20% are set as testing samples.

2) Training Details:

Step 1: Log Parsing

- Acquire structured logs, first-letter log event template E_M of 410 events ($M = 410$) and following occurrence matrix $O \in \mathcal{R}^{M \times M}$ by LKE [6].

Step2: Window Representation

- Event Ordering
  Obtain highly-coupled event template by ACO, with ant number as 500, $\rho = 0.1$, $\alpha = 1$, $\beta = 2$ and $Q = 5$.
- 2D Window Representation
  Split whole dataset into samples by sliding windows that window span is set as T and step size as $\frac{T}{2}$.
  Divide each period T into 12 subwindows ($N = 12$) and represent it in a 2D form according to highly-coupled event sequence S. Thus behavior in each period is expressed in a $[12 \times 410]$ 2D window matrix.

Step3: Anomaly Detection

- Feature Extraction
  Compress each 2D window matrix into a $[1 \times 41]$ feature ($L = 41$). Each parallel module is designed to share 10 on width($W = 10$) and possess 32 channels. This step is implemented on a piece of Titan X(Pascal) GPU with Ubuntu 16.04.3 system.
- Binary-class Classification
  Classify the outliers and normal points by linear SVM in scikit-learn 0.19.1 package, with $tol = 1e-4$, $c = 1$ and penalty type as $'l1'$. As linear SVM outperforms SVM with non-linear kernels, we only discuss linear SVM in our experiments.

3) Evaluation Metrics:

On perspective of anomaly detection, the minor class of outliers is treated as positive tag, while normal points are labeled as negative samples. Generally, there is trade off between precision and recall. Thus, F-score is regarded as the balanced index in our problem.

$$\begin{aligned} accuracy &= \frac{TN + TP}{TN + TP + FN + FP}; \\ precision &= \frac{TP}{TP + FP}; \\ recall &= \frac{TP}{TP + FN}; \\ F\text{-}score &= \frac{2 \times precision \times recall}{precision + recall} \end{aligned} \tag{7}$$

B. Results Discussion

1) Results among Different Sliding Windows:

We first compare our method to the traditional method with linear SVM classifier refered in [5] on 6 different sliding windows. For multi-module CAE, we list the winner-take-all rate which makes contributions to the highest F-score in Table I by searching grid $\{10\%, 20\%, 25\%, 30\%, 35\%, 40\%, 45\%, 50\%, 60\%\}$. Both linear SVM classifiers implemented by scikit-learn package are configured with identical parameters that $tol = 1e-4$, $c = 1$ and penalty type set as $'l1'$.

From Table I, it is apparent that our proposed method outperforms the traditional linear SVM classification with 5% to 10% increase in F-score. The 1h/0.5h sliding window is with the lowest F-score compared to other sliding windows. And there is a trend that when the window size increases, the F-score rises up as well. The cause to this phenomena might be express by that the working tasks within small time intervals are more diversified, on the contrast, a longer interval might cover a working cycle of the system whose patterns are easier to recognize.

2) Comparison between Multi-module and Single Module: To evaluate the influences of the parallel modules in CAE, we conduct 4 simulations by changing the module construction in CAE. The 3 parallel modules in Fig 5 are assigned individually to an independent simulation. As the total channels in multi-module CAE are 96 after deep concatenation, the channels of single module CAE are set as 96 but not 32 to ensure the comparability. The hyper parameter winner-take-all rate $k\%$ is set to 40%, where most windows could achieve best result.

As demonstrated in Fig 6, the multi-module CAE achieves best F-score than other single module structures among 5 sliding windows, except the 3h/1.5h sliding window. For this exception window, the optimal F-score is achieved by the single module 2 of 78.40%, 1.2% higher than the multi-module. This result is acceptable since the optimal winner-take-all rate of this window is 30% for multi-module construction which achieves 80.37% in F-score.

3) Improvement by ACO Event Ordering:

To evaluate the effectiveness of event ordering step, we compare 2 models in Fig 7 that one model omits the event ordering which uses first-letter event template, while the other executes which applies highly-coupled event template. The configuration of multi-module CAE keeps the same, with winner-take-all rate $k\%$ set as 40%.

From Fig 7, we can observe that both models obtain higher F-scores than the simple linear SVM. And the ACO-ordered sequence improves the performance to a large extent. There are mainly 2 reasons for the result with first letter order is higher than the baseline. Besides of the contributions of multi-module CAE, the first letter order actually contains a couple of neighboring events in close relationship as they share the same first word. For example, event pair "data read plb error" with "data write plb error" and event pair "debug wait enable" with "debug interrupt enable" commonly occur together.

TABLE I
RESULTS AMONG DIFFERENT SLIDING WINDOWS

Sliding Windows				linear SVM [5]				Ours				
Span	Step	anomalies	total samples	Accuracy	Precision	Recall	F-score	k%	Accuracy	Precision	Recall	F-score
1h	0.5h	1660	10300	92.62%	64.38%	65.58%	64.98%	40%	94.37%	71.81%	75.81%	73.76%
2h	1h	1388	5151	93.02%	96.18%	65.28%	77.78%	40%	93.31%	84.44%	78.76%	81.50%
3h	1.5h	1235	3434	90.25%	97.37%	63.43%	76.82%	30%	90.68%	86.75%	74.86%	80.37%
4h	2h	1111	2575	88.93%	100.00%	64.38%	78.33%	40%	91.26%	94.57%	76.25%	84.43%
5h	2.5h	1016	2060	86.65%	98.04%	65.36%	78.43%	45%	89.08%	95.76%	73.86%	83.39%
6h	3h	938	1716	84.30%	97.83%	63.38%	76.92%	35%	87.79%	93.10%	76.06%	83.72%

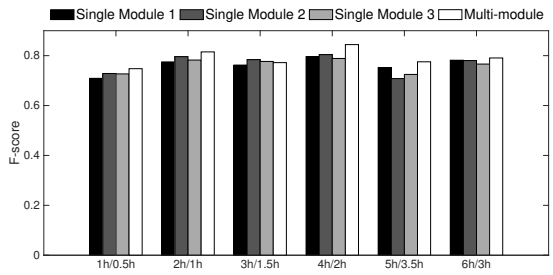


Fig. 6. Comparison between Muilti-module and Single Module

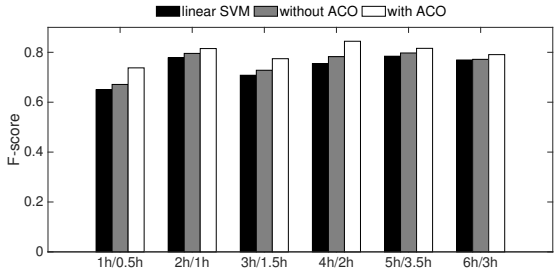


Fig. 7. Improvement by ACO Event Ordering

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a convolutional auto-encoder method for anomaly detection on system logs, taking advantage of time information under logs. The 2D window matrix representation based on an ACO-ordered event template sequence demonstrates the period behaviors of system in a detailed form. This expression takes time patterns into consideration while most previous work ignores. In addition, by introducing parallel modules in CAE, features among different time intervals could be integrated together as the latent space representation. Through the competitive results, we proved the effectiveness of our proposed method. In the near future, we wish to conquer some challenges in this field, such as visualization of the anomaly detection procedure and anomaly detection on log flow outliers with only normal dataset.

REFERENCES

[1] R. Charette and J. Romero. The staggering impact of it systems gone wrong. https://spectrum.ieee.org/static/the-staggering-impact-of-it-systems-gone-wrong.

[2] V Chandola, A Banerjee, and V Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[3] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 102–111. ACM, 2016.

[4] M. Basseville, I. V. Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs, 1993.

[5] S He, J Zhu, P He, and M. R. Lyu. Experience report: system log analysis for anomaly detection. In *ISSRE'16: the 27th IEEE International Symposium on Software Reliability Engineering*, pages 207–218. IEEE, 2016.

[6] Q Fu, J Lou, Y Wang, and J Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *ICDM'09: the 9th IEEE International Conference on Data Mining*, pages 149–158. IEEE, 2009.

[7] J. A. Hartigan and Manchek. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[8] H Mi, H Wang, Y Zhou, M. R. Lyu, and H Cai. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1245–1255, 2013.

[9] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder. Hourly analysis of a very large topically categorized web query log. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 321–328. ACM, 2004.

[10] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt. Debugging in the (very) large: ten years of implementation and experience. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 103–116. ACM, 2009.

[11] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

[12] Y Liang, Y Zhang, H Xiong, and R. Sahoo. Failure prediction in ibm bluegene/l event logs. In *ICDM'07: the 7th IEEE International Conference on Data Mining*, pages 583–588. IEEE, 2007.

[13] W Xu, L Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 117–132. ACM, 2009.

[14] J Lou, Q Fu, S Yang, Y Xu, and J Li. Mining invariants from console logs for system problem detection. In *USENIX Annual Technical Conference*, 2010.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[16] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

[17] A. Makhzani and B. J. Frey. Winner-take-all autoencoders. In *Advances in Neural Information Processing Systems*, pages 2791–2799, 2015.

[18] A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *DSN'07: the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 575–584. IEEE, 2007.