Application Performance Anomaly Detection with LSTM on Temporal Irregularities in Logs

Xavier Baril, Oihana Coustié, Josiane Mothe, Olivier Teste

Application Performance Anomaly Detection with LSTM on Temporal Irregularities in Logs

Xavier Baril
Oihana Coustié
xavier.baril@airbus.com
oihana.coustie@irit.fr
Airbus
Toulouse, France

Josiane Mothe
Olivier Teste
josiane.mothe@irit.fr
olivier.teste@irit.fr
ESPE, UT2J, Univ. de Toulouse, IRIT, UMR5505 CNRS
Toulouse, France

ABSTRACT

Performance anomalies are a core problem in modern information systems, that affects the execution of the hosted applications. The detection of these anomalies often relies on the analysis of the application execution logs. The current most effective approach is to detect samples that differ from a learnt nominal model. However, current methods often focus on detecting sequential anomalies in logs, neglecting the time elapsed between logs, which is a core component of the performance anomaly detection. In this paper, we develop a new model for performance anomaly detection that captures temporal deviations from the nominal model, by means of a sliding window data representation. This nominal model is trained by a Long Short-Term Memory neural network, which is appropriate to represent complex sequential dependencies. We assess the effectiveness of our model on both simulated and real datasets. We show that it is more robust to temporal variations than current state-of-the-art approaches, while remaining as effective.

CCS CONCEPTS

• **Software and its engineering** → *Software testing and debugging*; • **Theory of computation** → **Semi-supervised learning**; • **Computing methodologies** → **Anomaly detection**; **Neural networks**.

KEYWORDS

information system, event logs, anomaly detection

ACM Reference Format:
Xavier Baril, Oihana Coustié, Josiane Mothe, and Olivier Teste. 2020. Application Performance Anomaly Detection with LSTM on Temporal Irregularities in Logs. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3412157

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412157

1 INTRODUCTION

With the fast growing of their size and complexity, modern information technology systems contain numerous sources of potential faults and vulnerabilities [3, 6, 8]. The failure of their services can have significant consequences, ranging from degraded user's experience [11] to important financial losses [15]. The ability to accurately detect anomalies is therefore considered as a major issue [15]. Log analysis constitute a promising solution to system anomaly detection, since large-scale systems generate numerous execution logs [6, 11]. A *log* is a semi-structured message tracing the system execution [8]. It contains valuable monitoring information, such as the *timestamp* (time of occurrence) or the *content*, an often unstructured free text. The *event type* of a log describes the type of information being logged in the content part [1]. The *sequence_id* of a log is an optional field, gathering logs from a same sequence.

A *log anomaly* is an unexpected behaviour of the log data, and can be paired to a system anomaly. For instance, a late appearance of logs in a sequence may indicate a *performance anomaly* corresponding to an abnormal temporal irregularity in a service response. Hence, log anomaly detection is recognized as an efficient mean to perform system anomaly detection [3]. Manually analyzing large and complex log datasets represents a cumbersome and error-prone task [3, 8], justifying the need for automated data-driven solutions.

Supervised methods are the first historically used to detect anomalies in logs [5, 9]. They train binary classifiers, based on both normal and abnormal samples. Despite their encouraging results on balanced data, they suffer from the scarcity of abnormal samples in real datasets. In addition, they only detect anomalies matching the learnt abnormal class, preventing the detection of new anomalies. Unsupervised outlier detection [10, 14] overcome both the scarcity of anomalies, and the frequent unavailability of labels. These clustering techniques isolate the nominal samples in polluted datasets. However, they tend to simply detect samples that are significantly different from the majority regarding selected features [8].

Finally, novelty detection methods focus on detecting deviations from the nominal behaviour. Following a semi-supervised approach, a nominal behaviour model is learnt on anomaly-free data. New coming data are compared to this model and detected as anomalies if their behaviour does not match the nominal one. In these methods, logs are often represented as temporally-ordered sequences of event types. Fu *et al.* [7] train a Finite State Automaton to model these sequences as a unique workflow. Yet, the unique aspect of the model prevents the treatment of sequences that contain parallel tasks [6]. The DeepLog method [6] rather train a prediction task

with a Long-Short Term Memory (LSTM) neural network, which efficiently represents complex sequential dependencies based on multiple previous logs. DeepLog is regarded as one of the most up-to-date and accurate reference.

Nonetheless, representing the logs as sequences of event types raises challenging issues. Firstly, the sequences creation requires the knowledge of the *sequence_id*, which is not always available. Secondly, this representation discards the time-frame separating logs. This information is however key to define performance anomalies. The *performance anomaly detection* field focuses on detecting periods of slowdown or unavailability of a system or a service [2], and ranges from the detection of deny-of-service attacks [6] to the performance monitoring of cloud infrastructures [12]. For these anomalies, the sequential representation of the logs is insufficient: the time elapsed between logs must be integrated to the model in order to detect temporal irregularities in logs' appearance.

This paper focuses on the detection of performance anomalies and presents a new novelty detection approach named NoTIL, **No**velty detection based on **T**emporal **I**rregularities in **L**ogs. Based on the idea of counting the event types over time with a sliding window, NoTIL takes the time elapsed between logs into account by capturing the frequency of logs' appearance. NoTIL takes advantage of the LSTM ability to model complex temporal dependencies and applies it to model temporal correlations and detect temporal irregularities. In the following, section 2 details our NoTIL method. Section 3 evaluates its efficiency and section 4 concludes the paper.

2 NOTIL ANOMALY DETECTION METHOD

As a novelty detection method, NoTIL models the nominal behaviour of logs and detects anomalies as violations of the trained model. Especially, NoTIL trains a forecasting model and detects the samples with low prediction results as anomalies. In the context of performance anomaly detection, the prediction tasks and the data representation should be chosen so as to take into account the time elapsed between logs. Indeed, a performance anomaly is manifested within the logs as an abnormally long time-frame separating two consecutive logs. Contrary to the state-of-the-art methods, NoTIL represents the logs with a counting time window of the event types, taking into account the time elapsed between logs. We use an LSTM network for the prediction model, since they can model complex temporal relations [6, 11]. This section formalizes the problem of detecting performance anomalies before describing the data representation and the prediction model.

Notations. We use curly brackets $\{a_1, .., a_k\}$ to designate a set of k unique elements; $(a_1..a_k)$ denotes a sequence and $[\![a..b]\!]$ stands for the sequence of integers between a and b (included). $|E|$ is the number of elements of a set E and $\mathcal{P}(E)$ is its power set. The $floor$ function $\lfloor\ \rfloor$ is defined by: $\forall x \in \mathbb{R}, \lfloor x \rfloor = max\{m \in \mathbb{Z} \mid m \leq x\}$. We note $\wedge$ the logical AND between conditions. Finally, for a random variable X on an event space Ω, F_X is its cumulative distribution function. $\forall \alpha \in [\![0..1]\!]$, X_α is the α-upper critical value of the probability distribution of X, define by $F_X(X_\alpha) = 1 - \alpha$; $\forall x \in \Omega$, x is *significantly higher* than X, noted $x \gg X$, if $x > X_\alpha$.

Problem statement. Let L be a dataset of logs, occurring in a discrete time-frame $[\![1..T]\!]$, $T \in \mathbb{N}^+$. E is the set of possible event types in L. For a log $\ell \in L$, we note $e_\ell \in E$ its event type and $t_\ell \in [\![1..T]\!]$ its timestamp. For any $\ell \in L$ and $e \in E$, we define $\texttt{next}_e(\ell) \in L$ as the next closest log of type e: the first log of type e that happens after ℓ. We define $d_e(\ell) = t_{\texttt{next}_e(\ell)} - t_\ell$, the duration between ℓ and $\texttt{next}_e(\ell)$. We assume that the nominal value of such a duration is a random variable $D_{e_\ell e}$ following an unknown distribution. The actual duration $d_e(\ell)$ corresponds to a performance anomaly if $d_e(\ell) \gg D_{e_\ell e}$. We assume that anomalies concern instants $t \in [\![1..T]\!]$ and we define a *labelling strategy* f as a function indicating which instant(s) are labelled abnormal: $f : L \times E \rightarrow \mathcal{P}([\![1..T]\!])$. The performance anomaly detection problem consists in retrieving the set of temporal anomalies in L: $T_L^A = \{f(\ell, e) \mid \ell \in L,\ e \in E \wedge d_e(\ell) \gg D_{e_\ell e}\} \subset [\![1..T]\!]$.

Data representation. We define a *window size* $w \in \mathbb{N}^*$ and a temporal *window* W_i, with a beginning time t_{W_i} and an ending time $t_{W_i} + w$, with $t_{W_i} < t_{W_i} + w \leq T$. A log ℓ belongs to a window W_i iff $t_{W_i} \leq t_\ell < t_{W_i} + w$. We represent W_i by a *counting vector* $C_i \in \mathbb{N}^{|E|}$ that counts, for each event type $e \in E$, the number of occurrences of e in the window W_i. The j^{th} element of C_i is expressed as $C_i^{(j)} = \left|\left\{\ \ell \in L \ \middle|\ t_\ell \in [\![t_{W_i}..t_{W_i} + w]\!] \ \wedge\ e_\ell = e_j\ \right\}\right|$. We opt for a sliding window mechanism, and define s as the sliding offset and N the number of sliding windows is L.

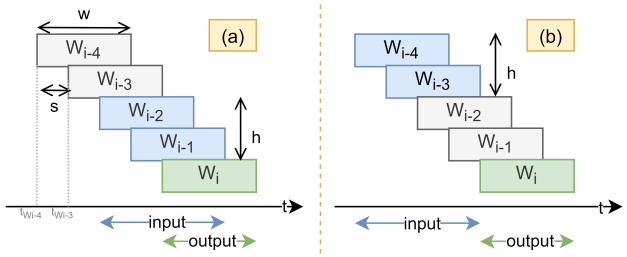

Figure 1: A set of $h = 2$ input windows of size $w = 6$ and shifted by $s = 2$ (blue) and its associated output (green) in an overlapping scenario (a) and a non-overlapping one (b)

The prediction task. As in Deeplog, we train a model for the task of forecasting the next window. We define a hyper-parameter h, called *look-back* [6] so that h consecutive windows are used to forecast the next window. Due to the sliding window mechanism, consecutive windows share common instants. To avoid the introduction of a bias between input and output — as depicted in the overlapping scenario in Figure 1(a) — we predict the output counting vector C_i with the input matrix M_i formed by the previous vectors $(C_{i-\alpha-h}...C_{i-\alpha})$, where $\alpha = \frac{w}{s} - 1$ (Figure 1(b)). As aforementioned, we opt for a LSTM model to train the prediction task. We formalize its role as a function LSTM that associates to an input matrix M_i a predicted output $\texttt{LSTM}(M_i)$. This predicted output is then compared to the real output C_i thanks to an error measure Δ, a function defined as $\Delta : \mathbb{N}^{|E|} \times \mathbb{N}^{|E|} \rightarrow \mathbb{R}^+$.

Finally, a threshold τ, learned during the validation, enables to split the windows into normal and abnormal subsets. The abnormal windows $W^{(A)}$ retrieved by NoTIL are defined as:

$$W^{(A)} = \left\{W_i \mid \alpha + h \leq i < N,\ \Delta\big(C_i, \texttt{LSTM}(M_i)\big) > \tau\right\}$$

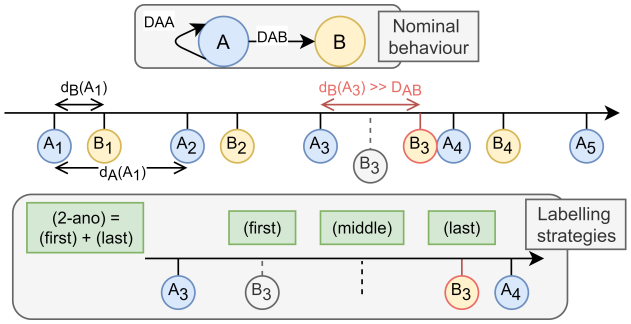

Figure 2: Generation scenario of Simu_binary. Top: nominal model. Center: temporal sequence with performance anomaly in A_3 - B_3. Bottom: labelling strategies for the anomaly.

To compare this set to $T_L^{(A)}$, the set of actual temporal anomalies of L, we transpose this latter into a set of abnormal windows:

$$W_{real}^{(A)} = \{W_i \mid \alpha + h \leq i < N \ \wedge \ \exists\, t \in T_L^{(A)}, t_{W_i} \leq t \leq t_{W_i} + w\}$$

3 EVALUATION

We assess the accuracy of NoTIL on different scenarios and compare it with that of DeepLog [6]. We present the evaluation framework before commenting the experimental results.

Data presentation. We evaluate our proposition on both simulated and real-world datasets. We first generate an artificial dataset **Simu_binary** to study the behaviour of the methods in a minimal configuration. As depicted in Figure 2, **Simu_binary** contains 2 event types A and B. D_{AA} and D_{AB} represent the nominal durations between consecutive logs of the corresponding event types. We inject performance anomalies in some transitions between A and B: in Figure 2, the transition between A_3 and B_3 is abnormal since $d_B(A_3) \gg D_{AB}$. D_{AA} and D_{AB} are Gaussian distributions with low variances (for steady scenarios). We generate several scenarios by changing the mean values. We then generate scenarios with higher variances to study the robustness of the methods to temporal irregularity. We also generate **Simu_nevents**, with new contextual event types that happen regularly, independently from one another and from the sequence of interest. This context matches better a real situation, where numerous types of logs are generated in parallel. In these artificial scenarios, we simulate a time-frame of $T = 10000$ for each of the training, validation and testing sets, and inject 1% of anomalies in the validation and testing sets.

We finally experiment on a real dataset. **Openstack** logs monitors the creation and use of virtual machine (VM) instances. The authors of DeepLog generated and made available their own **Openstack** dataset with injected performance anomalies. The dataset contains 47 event types, among which 23 are implied in a sequence (a VM instance). 4 sequences contain temporal anomalies, in the same transition time, for a global rate of 0.7% of abnormal sequences.

State-of-the-art methods selection. We study the behaviour of DeepLog towards performance anomalies. DeepLog proposes two independent anomaly detectors: we note SDL its sequence anomaly detector, and PDL its performance anomaly detector. SDL only considers the sequence of event types, and is not specifically designed for performance anomalies. PDL processes each event type individually and analyses the time elapsed between consecutive logs. PDL is therefore eligible for performance anomaly detection: theoretically, it can catch temporal irregularities, but might struggle to detect abnormal correlations between logs of different event types. We also study the reference state-of-the-art unsupervised method noted PCA [14], in order to prove the superiority of the novelty approach.

Labelling strategy. For the simulated data, we propose different labelling strategies, defining the aforementioned function f and graphically represented in the example of Figure 2. In these strategies, the abnormal instants $f(A_3, B)$ are:

- (**first**) the time when the log should have occurred, $t_{A_3} + D_{AB}$,
- (**last**) the time when the delayed log occurred, t_{B_3},
- (**2-ano**) both,
- (**middle**) the middle instant between both, $\frac{t_{B_3} + t_{A_3} + D_{AB}}{2}$.

We study the impact of the labelling strategies on the results of the understudied methods in order to select a strategy that does not introduce any bias.

Implementation considerations. DeepLog and NoTIL are implemented with Pytorch framework and with the same hyperparameters provided in [13]. For our hyperparameters, we set $s = 1$ to observe all the possible window combinations and experimentally determined the w and h values at $w = D_{AB} - d_B(A_3)$ (the difference between expected and observed duration) and $h = 2 \times w$. PCA is not based on a prediction task, so the detection should be performed on a window size that covers both the input and the output of the other methods. Hence, we chose a window size of $w_{PCA} = w + h \times s$.

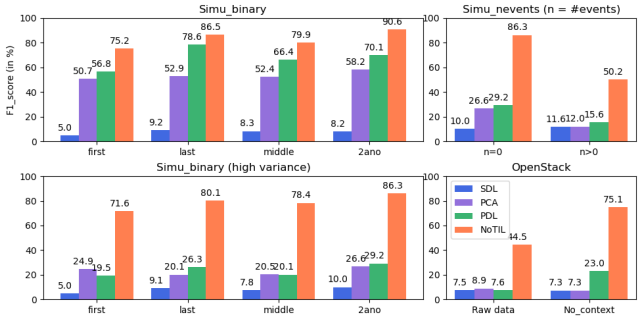

Figure 3: F1_scores of the understudied methods on simulated datasets and OpenStack.

Evaluation measures. Measuring the accuracy of an anomaly detector in our context consists in comparing the aforementioned sets $W_{real}^{(A)}$, the real abnormal windows and $W^{(A)}$, the predicted ones. The traditional F1-measure is not suitable for the performance anomaly context, since it does not tolerate any temporal delay between detected and real anomalies. Hence, we refine the definition of false positive and negative: for $q \in \mathbb{N}$, a real (resp. predicted) anomaly in W_i is a false negative (resp. positive) if no detected (resp. real) anomaly is observable within the sequence

$(W_{i-q}...W_{i+q})$, otherwise it is a true positive. We chose $q = \lfloor \frac{D_{AA}}{2} \rfloor$, in order to detect anomalies at a sequence-level temporal precision.

As most of the novelty detection methods, NoTIL learns a threshold to distinguish abnormal and normal samples. This threshold is learnt during a validation phase and is a lever to balance false positive and false negative. This step can be used to avoid detecting nominal outliers: novelty detection can detect anomalies that are more frequent than some nominal outliers, as long as they were not seen during the training phase.

Detection results. Figure 3 presents the F1-scores of the methods for the aforementioned datasets. For all the simulated data, we generated multiple scenarios — with different D_{AA} and D_{AB} values — and calculated the average score for each understudied method.

In the noise-free version of **Simu_binary**, both PDL and NoTIL present overall good results, with a slight overall advantage for NoTIL. Unsurprisingly, PDL benefits from the (**last**) labelling strategy, since it detects anomalies from the time elapsed since the previous log. B_3 and B_4 carry high anomaly scores since B_3 is abnormally far from B_2 and close to B_4. NoTIL also detects two abnormal instants, yet, these instants are aligned with the (**2-ano**) labels. The SDL proposal shows very disappointing results since the understudied anomalies have no signature at a sequence level, and are not observable if time is neglected. PCA method shows mitigate results, mainly due to the temporal imprecision necessary to encompass both the input and the output in the studied window.

The noisy version results highlight the limit of the PDL's representation of time. Since the occurrences of A are not regular, neither are those of B. Hence, the method struggles in learning the nominal behaviour of the event type B, and fails to detect the actual anomalies. PCA also collapses: the multitude of nominal samples induces a lack of distinction between rare nominal profiles and anomalies. NoTIL is only slightly impacted by the irregularity of this scenario and still presents reliable results. Here, (**2-ano**) is unanimously the most advantageous labelling strategy: it is bias-free to select it for the next evaluations.

The experiments on **Simu_nevents** show an impact of the contextual framework on all methods: the already-low results of the state-of-the-art methods collapse, while NoTIL remains the only reliable solution, despite a fair decrease.

Finally, PDL presents very disappointing results in the raw **OpenStack** dataset (column *Raw data*): it tries to find a common forecasting error threshold, whatever the event type is. Since half of the event types are not implied in a sequence (contextual logs) and do not bear any signature of the anomalies, applying a threshold on these event types leads to numerous false positives. To confirm this hypothesis, we propose an alternative version of the dataset (column *No_context*), which only contains the 23 event types involved in a sequence execution. This process is not innocuous: it requires domain knowledge to identify the *sequence_id* and eliminates 61% of the logs. Yet, it is not sufficient: an event type occurs twice in each instance, hence why there are two nominal durations between consecutive occurrences: (a) a short one, between occurrences of the same instance, (b) a much longer one between distinct instances. PDL fails in learning this nominal behaviour and triggers many false positives. On the contrary, NoTIL results are enhanced by this simplification process. Our analysis of NoTIL results however highlighted some false positive cases, that seem reasonable. For instance, a rare event type appears at an unusually high frequency in the testing dataset which is detected as abnormal by NoTIL.

4 CONCLUSION

We present a new method to perform anomaly detection, while maintaining the quantitative aspect of time, using a count of event types over time. NoTIL efficiently detects temporal irregularities in logs' appearance, which can be the signatures of performance anomalies. Our evaluations show a much more robust behaviour towards nominal temporal fluctuations than current approaches such as DeepLog. As a future work, we plan to extend our evaluation to a larger set of anomaly types. We also aim at extending the anomaly detection to parameters anomalies: logs contain variable parts that can be converted into numerical time series. We will also experiment other deep learning algorithms, with a focus on attention mechanisms [4, 11] in order to enhance both the computational performance and the interpretability of the model.

REFERENCES

[1] Xavier Baril, Oihana Coustié, Josiane Mothe, and Olivier Teste. 2020. METING: A Robust Log Parser Based on Frequent n-Gram Mining. IEEE The International Conference on Web Services (ICWS) (2020).

[2] Stéphane Bonnevay, Jairo Cugliari, and Victoria Granger. 2019. Predictive Maintenance from Event Logs Using Wavelet-Based Features: An Industrial Application. In 14th Inter. Conf. on Soft Computing Models in Industrial and Environmental Applications (SOCO) (Advances in Intelligent Systems and Computing, Vol. 950). 132–141.

[3] Andrea Borghesi, Antonio Libri, Luca Benini, and Andrea Bartolini. 2019. Online Anomaly Detection in HPC Systems. In IEEE Inter. Conf. on Artificial Intelligence Circuits and Systems, AICAS. 229–233.

[4] Andy Brown, Aaron Tuor, Brian Hutchinson, and Nicole Nichols. 2018. Recurrent Neural Network Attention Mechanisms for Interpretable System Log Anomaly Detection. CoRR abs/1803.04967 (2018). arXiv:1803.04967

[5] Mike Y. Chen, Alice X. Zheng, Jim Lloyd, Michael I. Jordan, and Eric A. Brewer. 2004. Failure Diagnosis Using Decision Trees. In 1st Inter. Conf. on Autonomic Computing (ICAC). 36–43.

[6] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In Proc. of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS. 1285–1298.

[7] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In 9th IEEE Inter. Conf. on data mining. 149–158.

[8] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2016. Experience Report: System Log Analysis for Anomaly Detection. In 27th IEEE International Symposium on Software Reliability Engineering, ISSRE. 207–218.

[9] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra K. Sahoo. 2007. Failure Prediction in IBM BlueGene/L Event Logs. In Proc. of the 7th IEEE Inter. Conf. on Data Mining (ICDM 2007). 583–588.

[10] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining Invariants from Console Logs for System Problem Detection. In USENIX Annual Technical Conference, 2010, Paul Barham and Timothy Roscoe (Eds.).

[11] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, and Rong Zhou. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. In Proc. of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI, Sarit Kraus (Ed.). 4739–4745.

[12] Yongmin Tan, Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Chitra Venkatramani, and Deepak Rajan. 2012. PREPARE: Predictive Performance Anomaly Prevention for Virtualized Cloud Systems. In IEEE 32nd Inter. Conf. on Distributed Computing Systems. 285–294.

[13] Yifan Wu. 2018. DeepLog. https://github.com/wuyifan18/DeepLog.

[14] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Largescale system problem detection by mining console logs. (2009).

[15] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. 2016. Automated IT system failure prediction: A deep learning approach. In IEEE Inter. Conf. on Big Data,. 1291–1300.