Automated IT System Failure Prediction: A Deep Learning Approach

Ke Zhang*, Jianwu Xu†, Martin Renqiang Min†, Guofei Jiang†, Konstantinos Pelechrinis* and Hui Zhang†
* *School of Information Sciences, University of Pittsburgh*
† *NEC Laboratories America*
* {*kez11, kpele*}*@pitt.edu*, † {*jianwu, renqiang, gfj, huizhang*}*@nec-labs.com*

***Abstract*—In mission critical IT services, system failure prediction becomes increasingly important; it prevents unexpected system downtime, and assures service reliability for end users. While operational console logs record rich and descriptive information on the health status of those IT systems, existing system management technologies mostly use them in a labor-intensive forensics approach, i.e., identifying what went wrong after the fact. Recent efforts on log-based system management take an automation approach with text mining techniques, such as term frequency - inverse document frequency (TF-IDF). However, those techniques lead to a high-dimensional feature space, and are not easily generalizable to heterogeneous log formats.**

In this paper, we present a novel system that automatically parses streamed console logs and detects early warning signals for IT system failure prediction. In particular, our solution includes a log pattern extraction method by clustering together logs with *similar* format and content. We then resemble the TF-IDF idea by considering each *pattern* as a word and the set of patterns in each discretized epoch as a document. This leads to a feature space with significantly lower dimensionality that can provide robust signals for the status of the system. As system failures tend to occur very rare, we apply a recurrent neural network, namely, Long Short-Term Memory (LSTM), to deal with the "rarity" of labeled data in the training process. LSTM is able to capture the long-range dependency across sequences, therefore outperforms traditional supervised learning methods in our application domain. We evaluated and compared our proposed technology with state-of-the-art machine learning approaches using real log traces from two large enterprise systems. The results showed the advantage and potentials of our system in prediction of complex IT failures. To our knowledge, our work is the first that employs LSTM for log-based system failure prediction.

***Keywords*-System Management; Failure Prediction; Log Analysis; Text Mining; Recurrent Neural Network; LSTM; Deep Learning;**

I. Introduction

With recent developments in computing, we have been witnessing rapidly increasing levels of reliance on computing technology for a variety of mission critical services. An enormous number of everyday life functions, ranging from banking systems to utility consumption control, are now relying on a series of heterogeneous computing systems. For example, with the vision of smart cities having started being realized, an increasing number of urban services (e.g., transportation, emergency and non-emergency response etc.) rely more on various IT systems. It should be evident that minimizing the downtime of these systems is extremely important and sometimes crucial even for the function of the society (e.g., imagine a scenario where the system controlling the subway routing and scheduling is down). There are numerous examples that have showcased the impact of a disruption from a computing failure. For example, in March 2008 a computing failure in the baggage system at Heathrow airport is estimated to have cost $32 million and affected 140,000 people [1].

Console logs record an IT system's operational states and events over time, and they carry rich and descriptive information. Relevant research on log analysis is focused on three distinct management problems; (i) forensic analysis, (ii) fault detection, and (iii) system failure prediction. Forensic analysis is a post-analysis of system logs and performs with the aim of identifying the root cause of the failure [2], [3]. Fault detection aims to quickly detect the signs of critical failures when they occur, and anomaly detection is a common methodology from the view of log analysis [4], [5]. On the contrary, failure prediction is a proactive approach, which aims at providing early warnings for potential failures.

One of the challenges with log-based failure prediction is the extremely large number of features required to achieve acceptable performance. For example, the current state-of-the-art approach [6] utilizes approximately 400 features extracted from the text of the system logs. This creates another problem with existing approaches related to the heterogeneity of computing systems and consequently their logs. To overcome both of the aforementioned issues, we develop a system that is built on new text mining and deep learning methods, and can be applied in heterogeneous logs; thus, it is generalizable and interoperable.

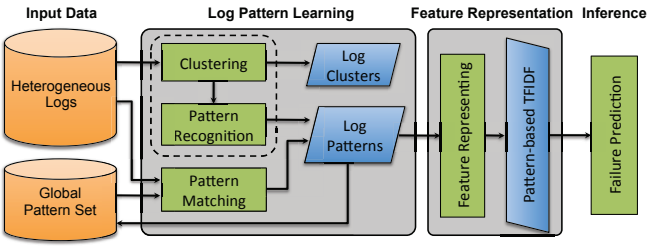


Figure 1. The framework of our solution

In a nutshell, our system extracts generic (non-textual)

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

features from the console logs that are passed through an LSTM network, which can be viewed as a deep neural network rolled in time, to perform failure prediction. As shown in Figure 1, our system first extracts format patterns from heterogeneous logs by clustering similar logs together and extracting the format/structure of log clusters (Section III-B). These patterns are then passed to the feature representation module, where sequential features over time are extracted. These patterns essentially substitute in the feature space the free text of the original logs, leading to a crucial and significant dimensionality reduction. The features over time are finally passed through an LSTM neural network to perform failure predictions (Section IV).

The main contributions of our work can be summarized as follows:

- We developed a general log analysis approach by learning regular patterns from heterogeneous logs, with their redundancy taken into account. Our method can significantly reduce the dimensionality and sparsity of feature space which are often the challenges associated with tranditional text mining techniques in log analysis.
- We formalized the failure prediction as a sequential classification problem, and we proposed a novel feature representation of coding logs during each time epoch in a low-dimensional vector space such that standard machine learning algorithms can be applied.
- We designed a prediction system based on a recurrent neural network (LSTM), which is able to outperform the current state-of-the-art systems.
- We evaluated our system with real system log traces from two enterprise systems, and validated its effectiveness through the performance metrics of the area under the curve of precision-recall, predictable time interval, and predictable frequency. We also showed the interpretability of our system for failure pre-diagnose.

The rest of the paper is organized as follows. Section II presents the failure prediction problem definition. Section III discusses the log analysis approach, as well as, the features used for the prediction. Section IV describes the LSTM-based prediction system, while Section V presents our evaluations. Section VI discusses the related work, and Section VII presents the conclusions.

II. Problem Definition

The failure prediction problem we study in this work is formally defined in what follows.

Problem 1 (Computing System Failure Prediction): Given a component (or a subsystem) of a computing system $\mathcal{K}$ and a collection of console logs $\mathcal{L}(\mathcal{K})$ from this component, infer the probability of a failure $\pi_f(W)$ occurring at this component within time window W.

In order to solve problem 1 we will treat it as a supervised learning problem and in particular, as a binary classification problem for predicting failure events within a time window W prior to the occurrence of failures. The training data for our supervised learning model consist of features extracted from the console logs of various computing systems (described in details in Section V) and failure labels provided by the system administrators. Note that problem 1 and consequently our solution is not tight to specific types of failures but it rather targets generic abnormal (failure) events due to either software or hardware problems.

The main goal of the prediction task is to successfully signal an alert before the failure occurs. Early detection is of particular interest in order to give the system administrator enough time to deal with the problem before it actually occurs. In our work, we consider similar requirements and metrics that have been considered in previous studies [6] and use the following definitions. In particular, the time of an early warning is important for the evaluation of the quality of predictive modeling and we define the following two periods that are further visualized in Figure 2 for clarity:

Predictive Period: A pre-defined time period right before a failure. An alert given in this period gives potential enough time for the administrator to act and is regarded successful. The time of the alert to the starting time of the failure is then the time window W defined in Problem 1.

Infected Period: A pre-defined time period right after a failure. When the component encounters a failure, it will most probably last until the problem is fixed and the component recovers back to its normal state. Therefore, data points within this period should not be used in the training or evaluation of the prediction model.

For the evaluation, true and false positives are defined as following:

True Positive refers to a predictive period within which at least one alert has been raised and is followed by an infected period.

False Positives refers to a predictive period within which at least one alert has been raised but is **not** followed by an infected period.

Our evaluations will further include more detailed metrics that capture the quality of a true alarm raised and are discussed in Section V.

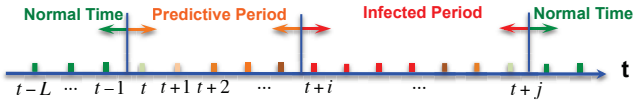


Figure 2. If we raise an alert during the predictive period - before the failure starts - the early prediction is considered a success.

In summary, the binary classification problem for failure prediction in IT systems takes as an input a sequence with length L of historical features $\mathbf{x}_{t-L+1}$, $\mathbf{x}_{t-2}$, $\mathbf{x}_{t-1}$, ..., $\mathbf{x}_t$, and the target $\mathbf{d}_t$ is binary vector taking two complementary values that represent negative examples in normal periods and positive examples in predictive period. The target can

be encoded in a variety of ways depending on the objective cost function in the model training phase.

If an alert is reported during the predictive period before the failure starts, then the prediction is considered as a success. The output from the learning model is essentially the probability of an alert, and an alert will be reported if such probability exceeds a pre-defined threshold ϵ.

III. Feature Extraction: Automated Pattern Recognition

A. Console Logs

Console logs are usually generated from a large amount of applications and/or functionalities, thus are heterogeneous with an extremely diverse and skewed word distribution. This brings challenges to extract compact and meaningful features from IT console logs using traditional text mining methods (e.g., topic modeling, bag of words etc.). However, console logs are usually generated from templates defined in the source code of applications, and thus have a pre-defined format. Also, logs are often redundant for the housekeeping purpose of applications. Finding a regular layout to represent a cluster of similar logs can help reduce the redundancy - not losing any important information at the same time - and summarize the meanings of log data.

Previous work [7] proposed to backtrack the source code to identify the regular layout of log data. However similar methods can only apply when the source code is available, and thus cannot easily generalize to a mixture of heterogeneous logs coming from various applications with different programming languages and logging styles. In this work, we provide a novel and general pattern learning method that can capture the log structure and the semantics of each log field.

B. Pattern Recognition

Given a set of console logs, a pattern is defined as the syntactic structure [1] of the log. Logs with similar format and content will align with the same pattern. Figure 3 shows some examples of plain logs and their pattern representation, where we can see there are clearly two clusters of logs, and each pattern essentially captures the common layout and content for a cluster of similar logs. In this section, we present the following steps to automatically recognize the syntactic structure of console logs.

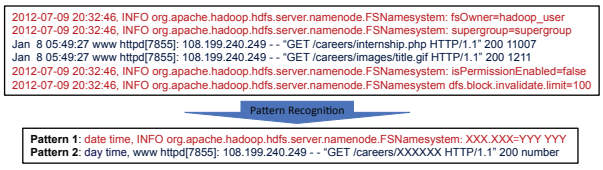


Figure 3. Logs are first clustered into groups and a regular expression based pattern is extracted for each group of logs.

[1]We will also use the words format and structure of a log record to refer to the layout interchangeably for the rest of this paper.

Log Tokenization and Time Stamp Standardization: The first data preprocessing step on heterogeneous logs is to tokenize the log data such that lower level information from the words or phrases of each log record can be identified and retrieved. However, heterogeneous logs from different applications and systems have different formats, and thus different tokenizers/delimiters. Without specific knowledge or human inspection, it will be easily unfair to use or not any pre-defined tokenizer for the entire set of heterogeneous log data. Thus, a very general delimiter should be used. The use of such delimiter should avoid interfering with the possible, even unknown, delimiters from the log data, or should it introduce confusion into the original data.

In our system, the single empty space is used as a general delimiter. All the words and special symbols except numbers will be separated by single empty space. Note that some units like IP addresses and URLs are all delimited by multiple single empty spaces. However, these special units will be recognized later as a whole.

Heterogeneous logs can have many different types of time stamp formats, which will make the following log clustering and pattern recognition difficult. Therefore, we detect all the time stamps within the logs and transform them into a standard format (YYYY / MM / DD HH : MM : SS . mss).

Log Clustering: Without domain knowledge with respect to the log formats, usage and sources, etc., a first step towards understanding and analyzing heterogeneous logs is to intuitively understand the syntactic structure of the log data. Towards that end clustering algorithms serve as a way to categorize data purely based on their intrinsic properties and relations. Thus, in our method, a clustering algorithm is applied on the heterogeneous logs so as to obtain an initial "view" of the data. More specifically, a hierarchical clustering algorithm is used to generate a hierarchical structure of the heterogeneous logs. Hierarchical clustering is preferred because (a) it can provide a range of granular views of the data (from coarse to fine-grained depending on the location we *cut* the dendrogram at), and (b) the data indexing and search in the following steps of our system is built up on a hierarchical tree structure for efficiency purposes. The hierarchical tree structure used in our method is denoted as Log Clustering Tree (LCT) [8].

In LCT, the hierarchical clustering algorithm Ordering Points To identify the Clustering Structure (OPTICS) [9] is implemented. OPTICS searches dense data regions by expanding from a certain data point towards all its neighboring data points that are sufficiently close under a pre-defined threshold. The clustering algorithm also generates a hierarchical clustering structure from the data point ordering, in which the denser data region within a looser data region, which is still qualified as a cluster, becomes a lower-level child of the looser region (i.e., cluster). Thus, the hierarchical structure constructed from OPTICS represents the inherent data relations among all the data points. There are two free

parameters in OPTICS, eps and $minpoints$. $minpoints$ parameter controls the minimum number of samples a valid cluster needs to contain, and eps specifies the maximum width of the cluster. These two parameters control the output of OPTICS algorithm.

Pattern Recognition: After clustering the log data, an overall structure of all the heterogeneous logs is generated. However, we still need to obtain more detailed patterns within each cluster.

Since within each cluster, the log records have very similar formats, pattern recognition is performed within a cluster using the idea of sequence alignment from Bioinformatics research. In particular, all the log records within a cluster are aligned together so as to identify the motifs (i.e, the most conserved and frequent common portion), and thus the common patterns, from the logs. Smith-Waterman algorithm [10] is used to perform pairwise alignment and then Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [11] strategy is used to perform multiple alignment. Note that the pattern recognition is done first in the leaf nodes, where the cluster purity is high and thus the alignment is clear, and then the pattern information is backpropagated from the leaves to the root node.

Pattern Matching: Once log patterns are generated from the previous step, the input heterogeneous logs are parsed against the patterns which are represented as regular expressions. Any input log will be either matched to any of the extracted log patterns (regular expressions), or not matched at all, which becomes an outlier. The output of the log pattern matching will be used for the feature representation.

C. Feature Representation

Once the input logs are parsed with the extracted patterns, each log is mapped into one pattern (except those logs which are not matched to any pattern). Thus instead of having a collection of plain logs during each epoch $[t, t + \Delta t]$ with epoch size Δt, we extract a collection of patterns p and count the frequency f_{p_i} of each pattern p_i. Choosing the epoch size Δt is not trivial and the objectives for choosing it include two contradictory requirements: (i) reducing the sparsity of feature representation, i.e., the percentage of epochs with no logs, and (ii) utilizing a smaller time granularity in order to make finer-grain predictions.

Pattern-based TF-IDF Features Extraction: In order to extract the appropriate features we borrow from TF-IDF which is often used as a feature representation of documents in information retrieval and text mining. In our case we treat each pattern p_i as a word, while the patterns of logs occurring in each epoch $[t, t + \Delta t]$ as a document e_t. The full set of documents/epochs E corresponds to the total observation period. Given a document e_t, the feature for the pattern p_i is defined as follows:

$$TFIDF(p_i, e_t, E) = TF(p_i, e_t) \cdot IDF(p_i, E) \quad (1)$$

$TF(p_i, e_t)$ is the term frequency of pattern p_i in a document/epoch e_t, and we utilize the logarithmically scaled frequency [12] given by:

$$TF(p_i, e_t) = \begin{cases} 1 + log(f_{p_i,e_t}) & , \quad f_{p_i,e_t} > 0 \\ 0 & , \quad f_{p_i,e_t} = 0 \end{cases} \quad (2)$$

where f_{p_i,e_t} counts the frequency of pattern p_i in document/epoch e_t. Finally, $IDF(p_i, E)$ is the inverse document frequency of pattern p_i, and using Laplace smoothing is given by:

$$IDF(p_i, E) = log \frac{|E|}{1 + |\{e_t \in E : f_{p_i,e_t} \neq 0\}|} \quad (3)$$

where $|E|$ is the total number of epochs and $|\{d_t \in E : f_{p_i,e_t} \neq 0\}|$ is the number of epochs where the pattern p_i appears.

IV. Failure Prediction

As defined in Problem 1, our prediction model outputs the probability of an upcoming failure, given the input sequence of historical feature vectors. If the computed probability exceeds a pre-defined threshold ϵ, an early warning is raised to signal a potential failure in the near future. The early warning sign is typically weak, and thus, hard to capture using simple models. Given the temporal dynamics of a computer system, the latter's states likely depend on a long historical trend. Traditional supervised learning methods, such as logistic regression, SVM and tree-based classifiers, simply consider an input sequence as independent features and are not able to capture the time-dependencies between them.

In order to overcome the above pitfall of traditional learning methods we will apply recurrent neural networks in our system. Recurrent Neural Network (RNN), which can be viewed as a deep neural network rolled in time, is a rich set of dynamic models that have been used for sequence generating and labelling. The internal state of the network does not only rely on the current input but it also depends on the states of the system during previous time. This is fundamentally different from the feed-forward neural networks. Nevertheless, classical RNNs are unable to store past input information for very long time [13], which diminishes its ability to model the long-range structure for the input sequence. LSTM is a RNN architecture designed to improve storing and accessing information compared to classical RNNs. LSTM has recently been successfully applied in a variety of sequence modeling tasks, including handwriting recognition, character generation and sentiment analysis [14]. Given the strong temporal dependencies in the computing system failure prediction aforementioned, in this work we apply an LSTM-based prediction network to model the dynamic nature of computer systems.

A. LSTM Prediction Network

Figure 4 presents the prediction architecture of the recurrent neural network we used in this paper. An input feature vector sequence $\mathbf{x} = (\mathbf{x}_{t-L+1}, ..., \mathbf{x}_t)$ with sequence length L [2] is passed to a stack of multiple recurrently connected hidden layers through weighted connections to compute the hidden vector sequences $\mathbf{h} = (\mathbf{h}_{t-L+1}, ..., \mathbf{h}_t)$ and consequently, the output vector sequence $\mathbf{y} = (\mathbf{y}_{t-L+1}, ..., \mathbf{y}_t)$. The output vector $\mathbf{y}_t$ can then be used to parameterize the probability distribution $Pr(d_t|y_t)$ of the target d_t.

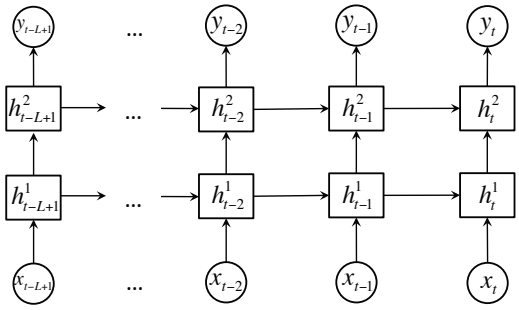


Figure 4. A many-to-many deep recurrent neural network prediction architecture. The rectangles represent the hidden layers, and the circles at the bottom and on the top represent the input layer and output layer, separately. The solid lines represent weighted connections.

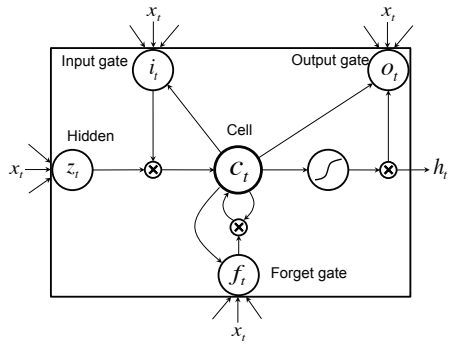


Figure 5. Long short-term memory cell

Unlike classical RNNs, LSTM tries to address the problem of long-term dependencies by introducing a purpose-built *memory cell* [15] to store information of previous time steps. Figure 5 shows the structure of a single LSTM memory cell. Access to memory cells is guarded by "input", "output" and "forget" gates. Information stored in memory cells is available to the LSTM for a much longer time than in a classical RNN, which allows the model to make more context-aware predictions. For our prediction task, we utilized the version of LSTM used in previous studies [16], [17], where the memory cell (hidden layer activation) is implemented by iterating the following composite functions:

[2]The sequence length L can be arbitrary long in order to memorize a very long historic sequence. However, there is a tradeoff with the computational resources and the time required to train the model.

$$\begin{cases} i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\ f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\ z_t = W_{xc}x_t + W_{cf}h_{t-1} + b_c \\ c_t = f_t c_{t-1} + i_t \cdot tanh(z_t) \\ o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\ h_t = o_t \cdot tanh(c_t) \end{cases} \quad (4)$$

where σ is the logistic sigmoid function, b_* are the bias terms and and i, f, o and c are the *input gate*, *forget gate*, *output gate* and *cell* vectors respectively, all of which have the same size as the hidden state vector h. The weight matrix W indicate the connections between gates, the cell, input and hidden states. For instances, W_{hi} is the hidden-input gate matrix, while W_{xf} is the input-forget gate matrix. The weight matrices from the cell to gate vectors (e.g., W_{ci}) are diagonal, so the m^{th} element in each gate vector only receives input from the m^{th} element of the cell vector.

Objective Function: To reiterate, we formalize the failure prediction problem as a binary classification problem, that is the target d_t is a binary vector with 2 complementary classes. The output y_t from the prediction network (Figure 4) is essentially a binary vector serving as a representation of the system status, which we can utilize to estimate the binomial distribution $Pr(d_t|y_t)$. Naturally, $Pr(d_t|y_t)$ can be parameterized by a softmax function at the output layer:

$$Pr(d_t = k|y_t) = \hat{y}_t^k = \frac{\exp(y_t^k)}{\sum_{k'=1}^{K} \exp(y_t^{k'})} \quad (5)$$

where K is the number of classes and therefore, in our case $K = 2$. For the objective function, we then utilize the binary cross-entropy cost function given the output at a time step t for one training example:

$$C = -\sum_{k=1}^{K} w_k \cdot [d_t^k \log(\hat{y}_t^k) + (1 - d_t^k) \log(1 - \hat{y}_t^k)] \quad (6)$$

where the target d_t^k is decoded as either 1 or 0. w_k is the weight for class k which is a particularly useful parameter when the target output is highly skewed (i.e., the training set is highly imbalanced with respect to the output class), where $\sum_{k=1}^{K} w_k = 1$.

V. Experiments

A. Dataset and Experiment Setup

Our dataset has been collected from two large enterprise systems, a web server cluster (WSC) and a mailer server cluster (MSC), as summarized in Table I. Each cluster is composed of multiple components with various types of applications running on them. The historical logs span 541 days for WSC and 161 days for MSC, where the system failures are recorded by the system administrators

whenever there is something wrong with a subsystem or component. The "failure" captures a variety of malfunctions and it can be translated as a subsystem or component that does not function properly or a subsystem/component that is completely down. In our dataset, we have 23 failure events for WSC and 12 for MSC, dispersing across the whole collection period.

We first discretize the time-series of historical logs and we resemble the idea of TF-IDF by taking all the logs in each epoch as a document. In particular, we select a smallest epoch size Δt that give sparsity, which is defined as the percentage of the epochs without any logs, less than 1%. At the same time we need a time granularity to be as small as possible in order to make a finer-grain prediction. Following this rule, we set $\Delta t = 10$ minutes for WSC and $\Delta t = 2$ minutes for MSC, which gives 0.817% and 0.091% sparsity accordingly. For model training and evaluation, we split the dataset into training and testing sets in time order, where the training set covers the first $2/3$ epochs in the history and the rest is used for testing. In particular, we have 12 of 23 historical failure events used for training in WSC dataset and 7 of 12 for MSC dataset. We then extract the patterns using all logs in the training set which gives us 111 and 183 patterns for WSC and MSC dataset, separately, where we set the parameter $eps = 0.1$ and $minpoints = 10$ in log clustering stage [3]. Given the extracted regular expression patterns, we then assign each log in the test set with a matched pattern. For the both two dataset, most of the logs (i.e., 81.4% for WSC and 96.7% for MSC dataset) can be found a matched pattern. Given the pattern representation for logs in each epoch (i.e., each document), we build the pattern-based TF-IDF feature vector as described earlier in Section III-C. Essentially, we transform a sequence of plain logs into a sequence of multiple, but with much lower dimensionality, feature vectors that we utilize to predict the system failure. Table II summarizes the main statistics of the data pre-processing and feature extraction.

Table I
TWO LOG DATASET FROM A WEB SERVER CLUSTER (WSC) AND MAIL SERVER CLUSTER (MSC).

Dataset	WSC	MSC
Collection Periods	[2013-02-24 - 2014-08-29]	[2014-03-22 - 2014-08-29]
# Logs	2,316,081	4,690,583

Note here that, at the training stage all the examples during the predictive period are considered as positive and those during the infected period will be discarded. A larger predictive period indicates we could expect an earlier warning identified before the failure occurs. However, if the predictive period is too large, the positive instances at earlier time might make it harder for our model to separate the true positive and negative examples. In our experiment, we utilize 3 hours for the predictive period, 8 hours for the infected period hence, allowing enough time to fix the problem. All these values were agreed upon with domain experts [4].

[3]We experiment with different values and our results are not sensitive to the parameter setting.

Table II
SUMMARY OF PATTERN LEARNING AND SEQUENTIAL PATTERN-BASED TF-IDF FEATURE VECTOR EXTRACTION.

Dataset	WSC	MSC
resolution (mins)	10	2
# patterns	111	183
# (%) Logs matched	1,885,022 (81.4%)	4,536,360 (96.7%)
# epochs/examples	77,696	23,048
sparsity	0.817%	0.091%

For our evaluations, we consider the following three performance metrics related to our early warning prediction problem.

PR-AUC: We consider the Area Under the Curve of Precision-Recall (PR-AUC) as our first metric. Given a predefined threshold ϵ and the definitions of true positives and false positives, we can evaluate the performance using the precision and the recall metrics:

- **Precision** = True Positive / (True Positive + False Positive)
- **Recall** = True Positive / (All positives)

By incrementally changing ϵ and repeating the process we calculate new pairs of precision and recall, and finally the precision-recall curve and the PR-AUC.

Predictable Interval: Another important performance metric we consider is how far ahead we can raise a true positive alarm, i.e., the earliest time during the predictive period when the model is able to report a correct alarm. Intuitively, the sign of a failure becomes stronger when approaching the time that the failure starts. The larger the time difference between the earliest reported warning and the starting time of the failure, the greater the chances that the system administrators will be able to diagnose and prevent the occurrence of the upcoming failure.

Predictable Frequency: We also consider the fraction of epochs during the predictive period that are predicted and reported as alarms. A higher frequency of alarms reported during the predictive period intuitively leads to a more confident decision of failure occurrence.

To calculate and evaluate the predictive interval and predictive frequency, we consider a threshold ϵ that gives the maximum recall when the precision is at least 0.7 [5]. We compare our system that is based on a deep learning

[4]We experimented with slight variations for the values of these parameters and did not find significant differences in the performance. Actually the average predictive interval is smaller than 3 hours as present in Table III.

[5]High precision is actually one important requirement for an automated reporting system, which provides a low false alarm rate

algorithm, i.e., LSTM, to state-of-the-art supervised learning methods, including logistic regression, SVM and random forest. The following are the parameters used for each model.

Logistic Regression: We use L2-regularized logistic regression, where we search linearly for the penalty parameter λ and report the one with the best performance.

SVM: We use the LIBSVM package [18] with linear kernel. We have tried other kernels as well (i.e., RBF) and the linear kernel performs the best on our dataset.

Random Forest: There are two important parameters for this model, namely, the number of trees and the number of randomly sampled variables to build a tree. We set the number of variables randomly sampled as the square root of the dimension of features, which is a typical empirical setting used in the literature [19]. We initially set a minimum value, i.e., 500, for the number of trees and use the Akaike information criterion (AIC) to search for the parameter.

LSTM: We begin by building a relatively small LSTM network with 2 hidden layers and 32 hidden units in each layer. Following previous work [20], we initialize all weight parameters uniformly in the range $[-0.08, 0.08]$, while initializing the LSTM forget gate with a slightly higher bias (set bias value to 1.0) to encourage remembering at the beginning. We then train the network using mini-batch stochastic gradient descent with batch size 5 and RMSProp [21] pre-parameter adaptive update with base learning rate 1×10^{-3} and decay factor 0.95. We experiment with different sequence length L by unfolding the network for L time steps. We train each model for 50 epochs and decay the base learning rate after 10 epochs [6] by multiplying it with the decay factor 0.95 for each additional epoch. We also set the weight bias in the loss function for the positive class w_2 as 0.99 ($w_1 = 1-w_2$) since our dataset is highly imbalanced. The parameter setting and training process work robustly for models with different sequence lengths.

B. Results

Figure 6 presents the curve of precision-recall for different sequence length L. Ideally we would like our system to have both high precision as well as high recall. This means that the part of the curves that are closer to the upper right corner are desirable. As we can see from these figures LSTM always provides a performance that is closer to the desired region and overall it outperforms the rest of the models. We can also observe that as one might have expected, increase the sequence length L leads to curves that are closer to the upper right corner. Figure 7 further presents the area under the curve of precision-recall (**PR-AUC**) for different sequence length L for both datasets.

As expected by the results above LSTM outperforms traditional logistic regression, SVM and random forest. It is also interesting to note that while increase L from 1 to 2 provides some benefit, longer sequence L does not further improve the baseline models. This can be attributed to the fact that these model assume independence between sequence of input features, therefore, not considering the dependencies between the points in the sequence. On the contrary, LSTM is able to capture the time dependency between feature time-series and its performance is monotonically improving with L when L is small.

Table III further presents the predictable interval and frequency for modeling with sequence length $L = 4$ where every algorithm has a relative good performance. We report the average values of all successfully predicted failure events. We have eliminated the results of logistic regression for both datasets and the result of SVM for MSC dataset since the recall is very low. We can see that LSTM can predict the failure much earlier than the baseline methods and can provide more confident (frequent) early alarms.

Table III
PREDICTABLE INTERVAL AND FREQUENCY WITH AT LEAST 70.0% PRECISION. LSTM CAN PREDICT EARLIER ON AVERAGE AND PROVIDE MORE CONFIDENT EARLY ALERTS.

Dataset		SVM	Random Forest	LSTM
WSC	Recall	72.7%	63.6%	90.9%
	Interval (mins)	64.3	45.5	73.0
	Frequency	51.3%	36.1%	66.2%
MSC -	Recall	–	60.0%	80.0%
	Interval (mins)	–	22.7	22.0
	Frequency	–	5.4%	30.4%

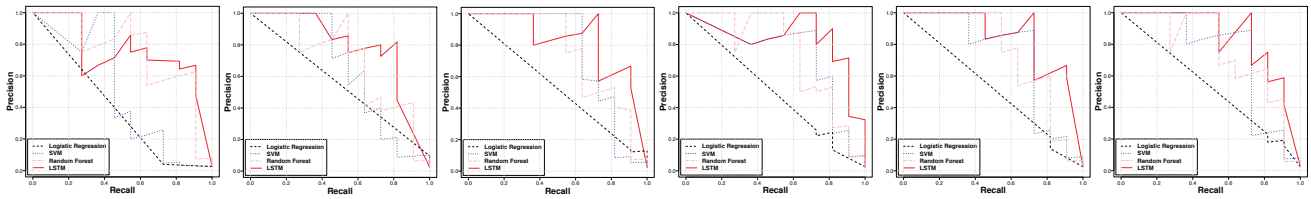
C. Parameter Evaluation for the LSTM Model

As the results shown above, LSTM outperforms traditional supervised learning models on our sequential classification task, even when we just apply a two-layer network with 32 hidden nodes for each layer. In this section, we further explore (i) the learning dynamics of LSTM training in our failure prediction task; (ii) the performance with longer sequence, i.e. $L > 6$, and (iii) whether the LSTM network can capture and track important features (patterns) for failure pre-diagnose.

Learning Dynamics: Figure 8 shows the learning curve of training and test performance with an increasing number of training epochs. As we can see, the testing performance reach the peak at around training epoch 20. Therefore, we do need to monitor the performance of LSTM on validation data to stop the training early to avoid model overfitting.

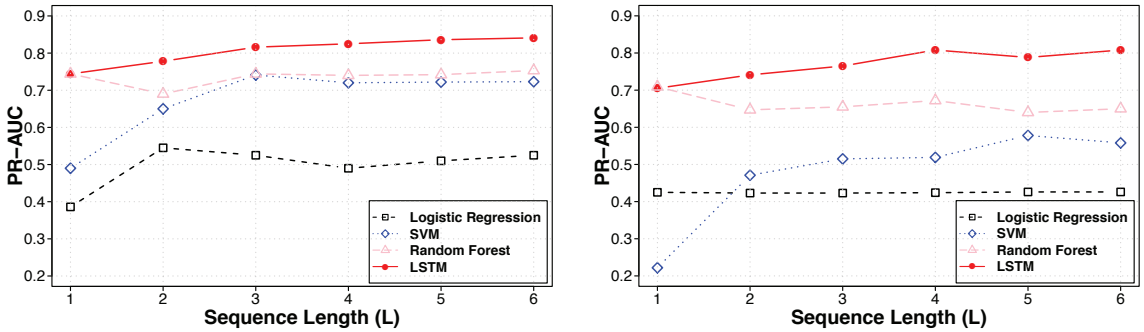
Longer Sequence: One interesting question is whether LSTM works better if we further increase the sequence length. Figure 9 shows the prediction performance with very long sequence lengths. We can see that very long sequences actually do not help improve the performance of the failure prediction in both two datasets (WSC and MSC). This might be due to the case that early warning signal might just be related to a few previous neighboring epochs. As we can

[6]Here, the epoch in neural network training is one full training cycle on the training set.



(a) $L = 1$ (b) $L = 2$ (c) $L = 3$ (d) $L = 4$ (e) $L = 5$ (f) $L = 6$

Figure 6. The curve of Precision-Recall for WSC dataset with regard to different sequence length L. The result for MSC dataset is eliminated here.



(a) WSC (b) MSC

Figure 7. The area under the curve of Precision-Recall (**PR-AUC**) for LSTM and other three baseline models using different sequence length L.

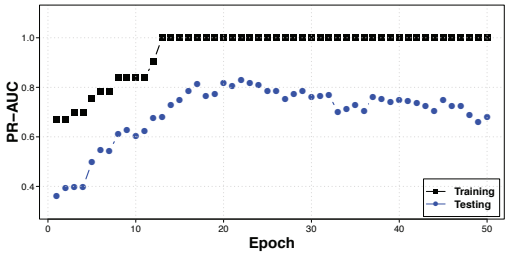


Figure 8. A typical learning curve of training and testing performance. Performing early stopping is helpful.

see, the size of average predictable interval, which is 73 minutes (around 8 epochs) for WSC and 22 minutes (around 11 epochs) for MSC, indicates that the early warning signs occur not very far from the failure starting time for our datasets.

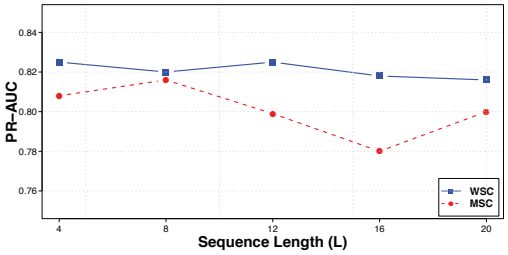


Figure 9. Very long sequence lengths do not help improve failure prediction performance in our dataset.

Critical Patterns Related to Failures: Applications in computing systems often have regular logging behaviors during the normal period, but rather alternatively behaves normally or abnormally when getting close to a failure time point. The applications often report some unusual logs (e.g., warning or error logs) with some specific log patterns during the predictive interval. These patterns play an important role in successfully predicting related failures and can serve as a summarization of logging behaviors for problem cause diagnose. In this part of our evaluations, we are interested in investigating what these important patterns are and how LSTM captures and tracks them for successful failure prediction. As an example, we train a network for WSC dataset with sequence length $L = 4$ and get the weights with the best performance. We consequently examine the weight connections between the input and hidden nodes. In particular, we build 2-hidden layer network with 32 hidden nodes, and then for each input feature (pattern) we examine the average weight of its connections to input, forget and output gates, which is then used to rank the input features/patterns.

Table IV presents the top 10 features with the highest average weight connections to different gates. We can see that there is a large intersection between the top ranked patterns with regard to different gate weight connections. In fact, the average weights for different gates are highly correlated, i.e., the spearman ranking correlation between input gate and forget gate is 0.63 (p-value < 0.001). We further compare the rank lists to the one selected by Random Forest, where we train the model with $L = 1$ and rank the patterns using

the feature Gini importance output from the model. We can see there is a large overlap between them. These identified log patterns correspond to meaningful signals that can help diagnose system failures, which indicate the capability of LSTM to accurately capture important input features.

Table IV
TOP 10 IMPORTANT FEATURES THAT ARE RANKED BY THE AVERAGE WEIGHT CONNECTIONS TO DIFFERENT GATES. THE BOTTOM ONE THE IMPORTANT PATTERNS SELECTED BY RANDOM FOREST.

Gates	Top 10 Patterns
Input	p_2, p_{79}, p_{51}, p_{38}, p_{59}, p_{126}, p_{135}, p_{54}, p_{149}, p_{136}
Forget	p_2, p_{126}, p_{51}, p_{59}, p_{38}, p_{136}, p_{83}, p_{115}, p_{79}, p_{45}
Output	p_{21}, p_{126}, p_{52}, p_{27}, p_{38}, p_2, p_{59}, p_{79}, p_{51}, p_{54}
RF	p_3, p_{38}, p_{81}, p_{79}, p_{80}, p_{150}, p_2, p_{127}, p_{84}, p_{126}

A real failure example: We further delve into the details of some failure events from our dataset and examine what the "important" features are. For example, we found a failure event as shown in Figure 10 that is most likely due to the network failure, where the server cannot respond requests in time. These failure events can be effectively captured by our model and revealed by some important patterns such as p_{79} and p_{38}.

< Event xmlns = \" http : / / schemas . microsoft . com / win / 2004 / 08 / events / event \" > < System > < Provider Name = \" MSExchange ADAccess \" / > < EventID Qualifiers = \" 16388 \" > 2070 < / EventID > **< Level > 4 < / Level >** < Task > 3 < / Task > < Keywords > 0x0080000000000000 < / Keywords > < TimeCreated SystemTime = **2014 / 08 / 18 08 : 22 : 55** / > < EventRecordID > 148040 < / EventRecordID > < Channel > Application < / Channel > < Computer > C1 .zzzz. xxx . co . jp < / Computer > < Security / > < / System > < EventData > < Data > [0] w3wp . exe ( ) [1] 6708 [2] g1. zzzz. xxx. co . jp [3] 70 [4] **Active directory response Since the limitation of the transmission timeout is exceeded, the operation was aborted** [5] SendTimeOut < / Data > < Binary > < / Binary > < / EventData > < / Event >

< Event xmlns = \" http : / / schemas . microsoft . com / win / 2004 / 08 / events / event \" > < System > < Provider Name = \" MSExchange Store Driver \" / > < EventID Qualifiers = \" 49156 \" > 1020 < / EventID > **< Level > 2 < / Level >** < Task > 1 < / Task > < Keywords > 0x0080000000000000 < / Keywords > < TimeCreated SystemTime = **2014 / 08 / 18 08 : 28 : 39** / > < EventRecordID > 89377 < / EventRecordID > < Channel > Application < / Channel > < Computer > C12. zzzz. xxx. co . jp < / Computer > < Security / > < / System > < EventData > < Data > [0] & quot ; Folder Content (g20 @ zzzz. xxx. co . jp) & quot ; [1] **You can not save changes to an item in the store**. < / Data > < Binary > < / Binary > < / EventData > < / Event >

Figure 10. A network failure event happened at 2014-08-18 09:00 in the WSC dataset. The early warning was reported more than 30 minutes before the failure which was captured by logs with pattern p_{79} (the upper one) and pattern p_{38} (the bottom one).

VI. RELATED WORK

In this section we briefly discuss literature related to our study.

Log-driven System Management: Console logs record an IT system's operational states and events over time. For effective system management, log analytics technologies have been developed - both in academia and in industry - and are mainly focused towards three directions related to fault management applications: forensic analysis, fault detection, and failure prediction.

Forensic analysis [2], [3] offers post-analysis of system logs with the goal of spotting the root of an observed problem. For example, Yuan *et al.* [2] analyze source code leveraging information provided by operational logs to infer what happened during the failed production run, while in [3] the authors develop a multi-purposed system that uses stack traces as its input and is able to diagnose a variety of application performance bugs ranging from problems in user functions to ones related with the system kernel.

Fault detection's goal is to quickly detect the signals for critical failures in order to prevent future problems. Anomaly detection is a common methodology used for fault detection. For example, Xu *et al.* [4] use the source code as a reference to parse console logs and construct system-related features on the parsed logs. The authors use these features as the input to a PCA model to perform anomaly detection. Kimural *et al.* [5] further developed a spatiotemporal factorization model to automatically learn underlying events from unstructured network log data. They then use the event models to detect complex and latent/hidden network problems.

Contrary to forensic analysis and fault detection, failure prediction is a proactive approach, and aims at providing early warnings for potential failures. For example, Sipos *et al.* [6] present a multiple-instance learning based approach for predicting medical equipment failures by mining equipment event logs that contain rich operational information. The approach was developed with active involvement from domain experts and utilizes approximately 400 features extracted from the event logs. Ma *et al.* [22] utilize the joint failure probability learnt from historical data to quantify and predict how likely a RAID group is expected to face multiple simultaneous disk failures, while Kimora *et al.* [23] adopt a supervised machine learning technique to associate network failures with the network log data that appeared before them using network trouble ticket data. To the best of our knowledge, our work is the first to combine deep learning with text mining techniques for log-driven failure prediction on large IT systems and we believe that it will be a new generation of systems based on deep learning.

Log clustering and pattern learning: There are mainly two approaches towards log clustering. The supervised learning method requires users first to manually label a set of log categories and use classifiers such as Naive Bayes to perform text categorization [24]. The other approach uses unsupervised learning for clustering, such as hierarchical partitioning process proposed in [25], and multi-pass data summarization method [26]. Once the cluster structure is obtained from the logs, string matching is used to extract the common patterns across multiple logs within the cluster.

Sequence modeling with deep neural networks: Recurrent Neural Networks such as Long Short-Term Memory (LSTM) are powerful models that are capable of learning effective feature representations of sequences when given enough training data. LSTMs have been successful in generating handwritten digit sequences with different styles [27], speech recognition [28], sequence-to-sequence machine translation [20], video-based action recognition [29] and more.

VII. Conclusion

In this paper, we present a log-driven failure prediction system for complex IT systems. The novelty of our work lies in the automated feature extraction of system states from streaming operational logs, and enabling earlier failure predictions through the LSTM approach on discovering the long-range structure in history data. Through the evaluation on real data traces, we showed our technology outperformed other state-of-the-art machine learning approaches in terms of PR-AUC, predictable interval and predictable frequency. In the future, we opt to investigate the online deep learning mechanism by dynamically updating the model with elapsed failure events.

References

[1] R. Charette and J. Romero, "The staggering impact of it systems gone wrong," *IEEE Spectrum*.

[2] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy, "Sherlog: Error diagnosis by connecting clues from runtime logs," *SIGARCH Comput. Archit. News*, vol. 38, no. 1, pp. 143–154, Mar. 2010.

[3] C. H. Kim, J. Rhee, H. Zhang, N. Arora, G. Jiang, X. Zhang, and D. Xu, "Introperf: Transparent context-sensitive multi-layer performance inference using system stack traces," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 235–247, Jun. 2014.

[4] 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*. ACM, 2009, pp. 117–132.

[5] T. Kimura, K. Ishibashi, T. Mori, H. Sawada, T. Toyono, K. Nishimatsu, A. Watanabe, A. Shimoda, and K. Shiomoto, "Spatio-temporal factorization of log data for understanding network events," in *2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014*, 2014, pp. 610–618.

[6] R. Sipos, D. Fradkin, F. Moerchen, and Z. Wang, "Log-based predictive maintenance," in *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2014, pp. 1867–1876.

[7] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *In Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM)*, 2013, pp. 118–126.

[8] X. Ning and G. Jiang, "HLAer: A system for heterogeneous log analysis," in *Proceedings of the SDM Workshop on Heterogeneous Learning*, 2014.

[9] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '99. New York, NY, USA: ACM, 1999, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/304182.304187

[10] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195 – 197, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022283681900875

[11] R. R. Sokal and C. D. Michener, "A statistical method for evaluating systematic relationships," *University of Kansas Scientific Bulletin*, vol. 28, pp. 1409–1438, 1958.

[12] C. D. Manning, P. Raghavan, and H. Schütze, "Scoring, term weighting, and the vector space model," 2008.

[13] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[14] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in neural information processing systems*, 2009, pp. 545–552.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *The Journal of Machine Learning Research*, vol. 3, pp. 115–143, 2003.

[17] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6645–6649.

[18] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[19] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[20] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[21] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio, "Rmsprop and equilibrated adaptive learning rates for non-convex optimization," *arXiv preprint arXiv:1502.04390*, 2015.

[22] A. Ma, F. Douglis, G. Lu, D. Sawyer, S. Chandra, and W. Hsu, "Raidshield: characterizing, monitoring, and proactively protecting against disk failures," in *Proceedings of the 13th USENIX Conference on File and Storage Technologies*. USENIX Association, 2015, pp. 241–256.

[23] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, "Proactive failure detection learning generation patterns of large-scale network logs," in *Network and Service Management (CNSM), 2015 11th International Conference on*, Nov 2015, pp. 8–14.

[24] T. Li, F. Liang, S. Ma, and W. Peng, "An integrated framework on mining logs files for computing system management," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 776–781.

[25] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 1255–1264.

[26] R. Vaarandi *et al.*, "A data clustering algorithm for mining patterns from event logs," in *Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM)*, 2003, pp. 119–126.

[27] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[28] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, 2014, pp. 1764–1772.

[29] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using lstms," *arXiv preprint arXiv:1502.04681*, 2015.