ANOMALY DETECTION AND DIAGNOSIS IN ASML EVENT LOG USING ATTENTIONAL LSTM NETWORK

ANOMALY DETECTION AND DIAGNOSIS IN ASML EVENT LOG USING ATTENTIONAL LSTM NETWORK

by

Xiaotong SHI

in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science,
at the Delft University of Technology,
Faculty of Electrical Engineering, Mathematics and Computer Science,
to be defended publicly on Wednesday October 30th, 2019 at 4:00 pm.

Student number:	4648226		
Project duration:	April, 2019 - October, 2019		
Thesis committee:	Dr.A.E.Zaidman	Full professor	SERG, TU Delft
	Dr.ir.S.E.Verwer(mentor)	Assistant professor	CYS, TU Delft
	Dr.J.van Gemert	Assistant professor	PRB, TU Delft
	Dr.M. Antonello(mentor)	Researcher/Developer	ASML

ABSTRACT

In the ASML test system, all activity events of the test are continuously recorded in event logs, and these logs are intended to help people diagnose the root cause of a failure. However, due to the large scale of the logs, manual inspection of these logs consumes lots of effort and time, and the lack of expert knowledge of engineers makes the efficient diagnosis more difficult. To improve the failure diagnosis efficiency in ASML, this paper proposes an attentional long-short term neural network into log sequence analysis. The LSTM neural network extracts the underlying dependencies in the event log and an attention layer is appended after to measure the importance of earlier events on the prediction of future events. The model learns the normal patterns from a large number of event logs from successful tests and detects deviations from normal patterns as anomalies. The likelihood of being abnormal of an event is measured by how far it deviates from the prediction. And the prediction process of the model can be understood by visualizing the attention scores of earlier events when the model makes decisions. Moreover, a visualization tool is built to illustrate the locations of anomalies and interpret the causes of anomalies through the attention maps.

PREFACE

I could not believe my student life is really coming to an end. Since I was a child, I have been told as a student, what is the right direction of life. Like other Chinese students, studying in a famous university was the only goal of my past two decades. I have been followed this default life path for so many years until now, when a brand new chapter is going to begin and nobody can tell me where to go. This is finally the time I force myself to think about what I really want for the rest of my life, even thought this freedom can be terrifying.

In the past few years, whenever I come cross a decision, I always choose the more difficult one. I chose to study abroad instead of working in a city I was familiar with; I chose the university of my dream, which was more difficult to graduate from and required more tuition than other options; I chose to write the master thesis with the topic I was really interested in at ASML instead of accepting the offer which could let me graduate earlier. All these choices have made my life more difficult, but also more rewarding. I am glad to find that I never regret these decisions.

I am very thankful to those who have always supported me whenever I felt uncertain about my decision, and who have given me generous compliment whenever I was not confident about myself. I would like to thank my mentor Mauro at ASML for giving me this opportunity to prove myself and support me all the time. Best wishes to hit lovely newborn twins. Thanks to Sicco, my supervisor at TUdelft, who gave me a lot of landmark advice when I got stuck in the work. Thanks to the weekly group meeting where I received great feedback from other master students. Thanks to EIT Digital for the scholarship, otherwise I would not be able to afford to study in Europe. Thanks to my boyfriend, who has patiently listened to my complaints and always been my side all these years. Finally, I would like to thank my parents, for their trust on my every decision and financial support, even though they never left China for their whole life.

I would not be the person I am today without the support of these people around me. Now, your girl needs to find out for herself the truth about life. The future may not be rosy, but it's time to say goodbye to the "me of the past."

Xiaotong Shi
Eindhoven, October 2019

Contents

1

INTRODUCTION

This chapter states the problem and introduces the dataset used in this paper. Several research questions are proposed and will be answered in chapter 6. At the end of this chapter, an overview of the structure of the paper is given.

1.1. PROBLEM STATEMENT

ASML produces complex machines. There are many reasons that could lead to a system breakdown of the machine, such as the inappropriate working environment of the machine, the logistic error in the software, or the wrong operation of the operator. Since testing on the real machine is too costly and risky, test engineers in ASML test software on the virtual machine called DevBench where tests can be submitted and executed. In order to help people understand the progress of the system, all the activity events are continuously recorded in event logs. The event logs can not only provide information when the system works correctly but also indicate when and where problems occur.

Hence, when a test fails, a test engineer can analyze the event log to diagnose the root cause of the failure. However, an event log usually contains thousands of events including hundreds of errors depending on the scale of the test, but only a few of them are important records and relevant to the root cause of the failure. Manually reviewing the logs consumes lots of effort and time and the lack of expert wafer scanners knowledge of engineers makes the efficient diagnosis more difficult. All of the above reasons cause that the time used in diagnosis is usually much longer than in problem-solving.

To help engineers diagnose failures, ASML designed a smart diagnostic tool (SDT) to automatically detect anomalies. However, due to the low accuracy of performance, this tool is rarely used by engineers in their works. Therefore, a smarter diagnostic tool is needed to help engineers save more time on solving problems.

This paper aims at applying the state-of-art deep learning algorithms on ASML event logs data with the purpose of root cause analysis. Given an event log of a failed test, the model is supposed to efficiently suggest the likely root cause of the failure. With the help of this tool, the user can save more time on diagnosing the failure compared to manually review the log or use SDT.

1

Some definitions are clarified to make the problem clearer:

- Anomaly
  There are many errors recorded in the event log, such as 1.1, but most of them present frequently both in the normal and abnormal test which means they are benign. Most such errors are caused by inaccuracies in the simulated test environments and some others are solved by automated recovery algorithms. In fact, only a few of the errors are crucial and likely to cause the breakdown of the system. So, this kind of dangerous errors are the anomalies we are interested in.
- The cause of the anomaly
  The cause of the anomaly refers to the event that is strongly relevant to the anomaly and has a contribution to the presence of the anomaly. The occurrence of this related event may indicate the occurrence of the anomaly in the future. In many cases, the last adjacent events are related to their adjacent events but not always, due to the interleaved process or the delay of logging.

1.2. DATA INTRODUCTION

Every test submitted and executed on the ASML testing platform generates an event log. The event log is generated to record the progress and mark the status when it occurs an error. Each logged event is a time-stamped event messenger which records when, where, why this event is logged. There are multiple processes in one test and some of them execute in parallel. Events in the same process are usually correlated to their adjacent events. However, due to the asynchronism of the logging system, it is possible that the events from different processes are not logged strictly in the order they generate.

The dataset we use in this paper is a collection of event logs of a test task. Each event log written in a plain text file has a set of chronologically ordered time-stamped events. The entire dataset is around 7GB and has 497 event log files in total including 468 event logs of successful tests and 29 event logs of failed tests which means the abnormal data takes up only 5.8% percent. The lack of positive class samples makes it is impossible to simply build a binary-classification model to classify normal and abnormal data.

Each event log in the dataset is labeled as TRUE (failed) or FALSE (successful) based on the final test result. In the event log of the failed test, not all of the events are abnormal. In fact, usually only a small part of events are relevant to the root cause of the failure. Moreover, the test can continue for several hours after an anomaly, flooding the event log with a mix of normal and abnormal events. Unfortunately, more specified labels for events are not available due to the huge amount of the data. The lack of true labels for events makes the precise evaluation at the event level very challenging.

On average, each event log has around 60000 events and each event belongs to an event type with variable parameters. There are 664 types of events in the entire dataset. An example of an event is shown in Figure 1.1 an Table 1.1.

```
07/24/2019  21:13:23.8095  Machine:FW40  (Rel:9.9.9.d, FTDS [26659], FTCI.c, ?.?, 744)
SYSTEM ERROR: FT-0114 DEFAULT
one or more tables are not closed yet
FTDS closed some databases that were left open.
```

Figure 1.1: An example of an event

Attribute	Example	Description
Timestamp	07/24/2019 21:13:23.8095	The stamped time at which the event occurred.This time is accurate to microseconds.
Machine ID	Machine:FW40	The ID of virtual machine.It is random generated but identical in one test.
Release version	Rel:9.9.9.d	The release version.
Process ID	FTDS[26659]	The ID of software component.It could be an executable or a python file.
File location	FTCI.c	The file location of software component.
Event type	SYSTEM ERROR	The severity of the event. It could be one of the following: EVENT, WARNING, ERROR, INFO.
Event code	FT-0114	The event code of the event. Each unique event code refers to a type of event. The first two characters of the event code identifies the software component cluster including the component that generated and/or logged the event.
Text description	One or more tables are not closed yet. FTDS closed some databases that were left open.	The text description of the event.The length is from one line to four lines.

Table 1.1: An example of an event

1.3. RESEARCH QUESTIONS

This paper aims to solve the following research questions.

RQ1: How to format the ASML unstructured log data and select the feature?

The log data is an unstructured plain text file as shown in the Figure 1.1 and Table 1.1. There is a lot of information recorded in the log but not all of them are useful. Finding an appropriate way to extract the most valuable information and abandon useless or misleading information is fundamental for building a good model.

RQ2: How to clean the ASML log data?

Since the average length of each event log is around 60000, data cleaning becomes extremely important in the efficiency and accuracy of the modeling. There are many unimportant events executed independently on tests. For example, there is a batch of events recording the initialization status and only presenting at the beginning of the test. And every test usually follows the same initialization steps. So these events are not inter-

esting to learn. Removing the events that are unlikely to be the root cause of the failure can help the model extract the most important pattern of the data and filter as many false positives as possible. However, I did not have any prior domain knowledge on recognizing the importance of the event and there is no official document in ASML listing useless events. So, cleaning the log data appropriately could be one of the challenges.

RQ3: The dataset has around 600 types of events and each log has around 60000 events on average. These events are known to be more or less causally related to their adjacent events. How to robustly model the causal dependencies among events in such a big and complicated dataset?

As the complexity of the software and the machine increases, there are more parallel processes in the test. Due to the asynchronism of the logging system, the event can not always be strictly logged at the time it generates. When two processes are executed in parallel, the events belonging to the first and the second process are interleaved in the log. On the other hand, when the interval time between two events is too short, the logging system cannot distinguish one from the other. The above reasons can cause the relative order of two events from two parallel processes becomes arbitrary.

RQ4: Given an event log file of a failed test, how to detect, localize and interpret anomalies?

Due to the lack of abnormal labels, it is impossible to apply a binary classification algorithm to classify the normal and abnormal data or build a model on the failures to learn the patterns of errors. So, it is challenging to build the model on such imbalanced data to distinguish the abnormal data from normal data and localize the anomalies. Also, besides the anomaly itself, it is interesting to know the cause of the anomaly as well. Compared to the statistical model such as the Bayesian network, the neural network model is usually a black box which has great performance on various tasks but lacks interpretability, so it is difficult to interpret the analysis process when the model gives the prediction. Is there a way to build an interpretable model while benefiting the impressive performance of the neural network?

1.4. SOLUTION

In this paper, I propose an interpretable deep learning model to learn the causal dependencies among events in the event log. A long-short term memory (LSTM) recurrent neural network is applied to model the sequence due to its impressive success on Natural Language Processing. To open the black box of LSTM, an attention layer is appended to measure the importance of each earlier event on the prediction of the future event. The model learns normal behavior from a large number of normal event logs. Given a new event log, the event which deviates from the model is considered to be more likely to be an anomaly. The cause of the anomaly can be interpreted by visualizing the attention distribution of the input sequence. The prior events with high attention scores are likely responsible for the occurrence of the anomaly.

1.5. CONTRIBUTIONS

There are following contributions in this paper:

- Effectively process and clean the ASML unstructured event log; design several noise filters to remove unimportant events from the event log.
- Apply Long-short term memory neural network on ASML event log sequence; introduce attention mechanism to interpret causally among events and understand the analysis process of the deep learning model. As far as I know, this paper is the first one applying the attentional LSTM prediction model on sequence anomaly detection and diagnosis task.
- Build a visual interface to help users intuitively detect, localize and understand the anomalies in the event log.

1.6. PAPER STRUCTURE

Chapter 1 introduces the problem and the goal of the paper, the ASML log dataset and research questions. Chapter 2 introduces the current approach used in ASML and summarizes related works from different aspects. Chapter 3 explores the dataset and introduces multiple ways to clean and compress the data. chapter 4 introduces the details of the proposed model. Chapter 5 experiments the model with different parameters and discussed the results. Chapter 6 answered the research question proposed in chapter 1 and chapter 7 is the conclusion and future work.

2

BACKGROUND

This chapter introduces the testing system and current diagnostic tool of ASML. Several related papers are reviewed from different aspects such as data pre-processing, feature selection and sequence modeling.

2.1. ASML TESTING SYSTEM

Testing is an essential part of the development chain in ASML. Testing on a real system can give you the best view of whether the software behaves correct, but it is too costly and takes too much time. Therefore, there is a smaller test platform TestArena provided via a simulated TwinScan called DevBench used primarily for software testing in ASML. The workflow of DevBench is shown in Figure 2.1. Test engineers can submit tests on the TestArena and create the DevBench where the tests can be executed.

Each test is an instance of a task. A task is a machine state or mode of operation that has a duration of typically minutes and is recognized by machine operators, Lot, Startup Drivers. Complex tasks can be split into multiple sequential phases or sub-tasks.

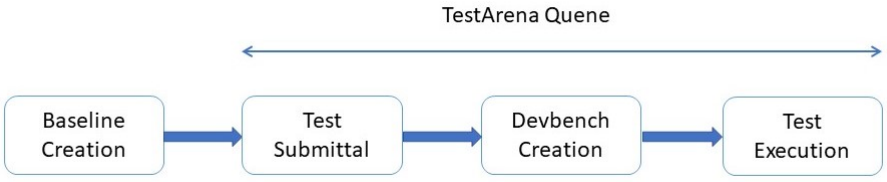


Figure 2.1: Testing flow on DevBench

2.2. ASML DIAGNOSTIC TOOLS

In order to improve the diagnosis efficiency, ASML designed some tools to automatically diagnose the event log of the test. Machine Indicated Root Error (MIRE) was the first tool of ASML but has been abandoned. Smart Diagnostic Tool (SDT) is the current diagnostic system widely used in software testing.

2.2.1. MACHINE INDICATED ROOT ERROR

Machine Indicated Root Error (MIRE) was designed to link errors and print the error trees as shown in Figure 2.2. It could link the root error and the following errors who are affected by the root error. However, these linkages usually can only indicate obvious relations between errors or solved errors which are usually irrelevant to the real root cause of the failure.

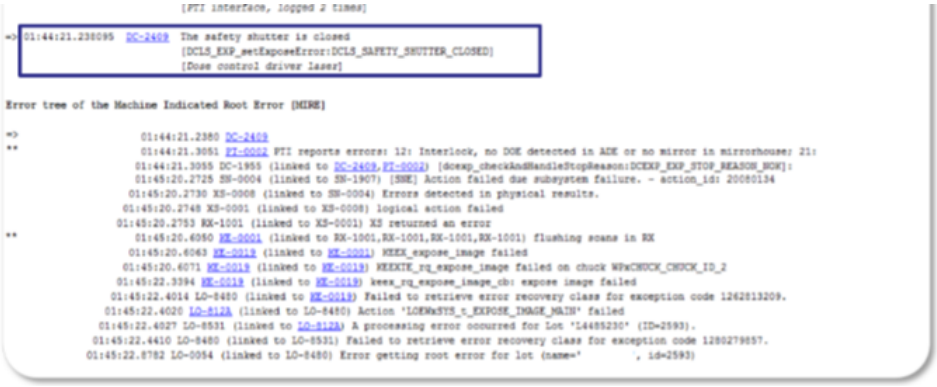


Figure 2.2: Machine Indicated Root Error

With the increasing complexity of ASML systems, MIRE became less and less effective. It often can not provide a quick and accurate pointer towards the cause of a failure. Manual inspection of all candidate root errors is still needed. Currently, MIRE has been abandoned and the link information provided by MIRE has also deprecated in the current ASML diagnostic system.

2.2.2. SMART DIAGNOSTICS TOOLS

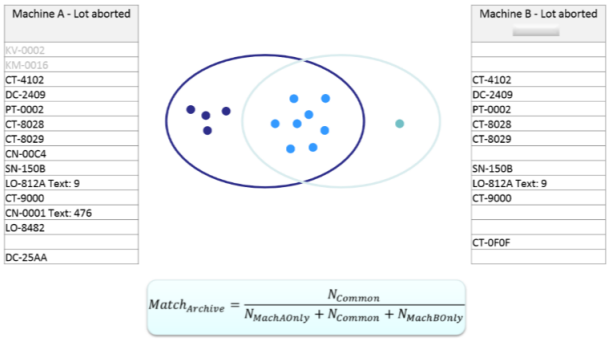
Considering the above problems, a new and more effective diagnostic method, Smart diagnostic tolls (SDT) was designed and is currently used in ASML testing systems. The core idea of SDT is to take into account all the relevant symptoms that are observed in a failed test. If the fault pattern of failure and its solution is known, then this information could be recorded and applied into future newly occurred failures.

Some definitions in SDT:

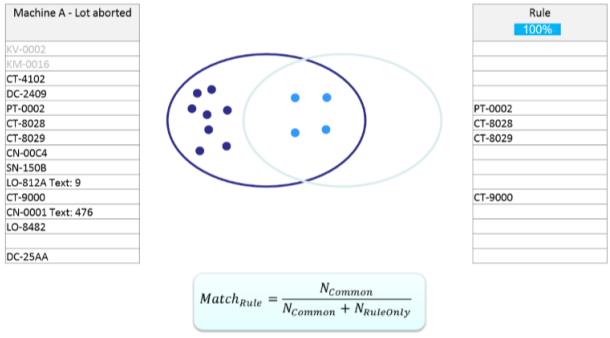
- Symptom: The symptom is derived from the information contained in a log entry. It can serve as a clue about what went wrong. Symptoms are similar to error codes.
- Fault pattern: A fault pattern is the collection of all relevant symptoms associated with a given failed task.

The idea would be explained by analogy with medical experience. Usually, each disease has a set of typical symptoms. For example, the patient who is suffering from the bleeding in his stomach can have symptoms such as spitting out blood and having a fever. However, having a fever can correspond to a wide range of causes. Relying on a single symptom to diagnostic might easily lead to a wrong treatment. Therefore, when examining a patient, a doctor will try to determine what the different symptoms are. When the symptoms are combined, the diagnosis becomes more accurate.

In SDT, the sets of symptoms of causes are collectively provided by users with diagnostic experience on different failures. The users can share the root error and relevant errors in the system when a root cause of failure has been found. When a new failure occurs, SDT will search its database for failures and their set of symptoms that occurred in the past, and compare the similarity between recorded fault patterns and new symptom lists. When the symptoms match, the cause of failure is identical.



(a) User similarity rule



(b) Expert similarity rule

Figure 2.3: The similarity rules of Smart diagnostics tools

Before the comparison, a noise filter is applied to remove symptoms that regularly occur in successful tasks. A ratio is defined as the percentage of presentation in success-

ful tasks. An event is filtered out when its ratio exceeds the threshold. The remaining set of error codes are used as fault patterns to compare with the data in the database.

For the fault patterns provided by users, the similarity is determined by the number of common symptoms and the number of different symptoms as shown in Figure 2.3(a). For the expert rule, the similarity checks how many percents of the symptoms specified in the expert rule as shown in Figure 2.3(b).

SDT allows the engineer to skip manual analysis of the machine event log and provides a direct recommendation with the most likely failure cause. Comparing to MIRE, diagnosis is based on multiple symptoms rather than relying on a single root error and the diagnostic result could be shared amongst users. However, there are still some drawbacks in SDT:

- This approach only considers the SYSTEM ERROR codes as symptoms. However, unusual SYSTEM EVENT/WARNING codes can also indicate an earlier abnormal behavior before the errors.
- This approach indeed releases the diagnostic engineers from tons of event entries. However, simply removing events which regularly present in successful tasks is not sufficient enough to remove as much useless/irrelevant events as possible. Manual effort is still needed to select fault patterns from remaining symptom lists. The accuracy and precision of this selection and the number of selected lists can directly affect the similarity calculation and further cause of failure identification.

2.3. RELATED WORKS

In this section, I summarize the related contents of some recent works from different aspects.

2.3.1. WINDOWING

An event log usually contains thousands of events. If we take each event instance as a feature, the dimension of the vector equals the number of events in the event log that it is impossible to feed such a high dimensional feature into the model. Therefore, there are some ways to separate the long sequence into several shorter sub-sequences. For the sake of computation simplicity and accuracy, the windowing approach is usually applied to slice up a long sequence into finite chunks. There are three major windowing approaches[Akidau *et al.*(2015)Akidau, Bradshaw, Chambers, Chernyak, Fernández-Moctezuma, La commonly used in temporal sequence processing.

- Fixed window
  The sequence is sliced by a static size window. This approach is suitable for the data logged at a fixed interval time, e.g. hourly or daily. If the data is periodic, using a fixed window can clear model the behavior in each interval time and compare the difference between periods.

- Sliding window
  The sequence is sliced with static window size and shifted to another window with the static step length. The step length may be shorter than the window length,

which means the windows are overlapped. The information of the events at the boundary of each window is kept in its adjacent shifted windows.

- Session window
  If there are multiple sessions in the sequence, the events can be grouped by a predefine session, e.g. process ID or host ID. The sequential order of events in each group can reveal the inner relationships in each session window. For example, grouping the pos machine trace by the customer ID could provide each customer's behavior information.

2.3.2. FEATURE SELECTION

The raw event log is usually an unstructured plain text file. Each event is text chuck including the information of timestamp, process ID, event code ID and so on. There are many ways to extract the most valuable information from these features.

- [Liang *et al.*(2007)Liang, Zhang, Xiong, and Sahoo] used the number of different types of events that occurred during a time window as features, including the number of INFO events, the number of WARNING events, the number of ERROR events. The number of these events would be accumulated over the entire observation period.
- [Chuah *et al.*(2010)Chuah, Kuo, Hiew, Tjhi, Lee, Hammond, Michalewicz, Hung, and Browne] counted the frequencies of each event and build a feature vector in the length of K (the size of event vocabulary) for each window. The frequency occurrence matrix is used to calculate the Pearson correlation coefficient. The strong statistical correlations indicate the occurrence of two events.
- [Lin *et al.*(2016)Lin, Zhang, Lou, Zhang, and Chen] vectorized the occurrence of the events in each time window in a binary vector in the length of K (the number of event types). When the frequencies of events are random in some degree, the appearance vector is more robust than the frequency vector. Using binary vector is faster when calculating similarity in the clustering task. However, considering the sparse characteristic, dimensionality reduction approaches such as PCA is needed to improve efficiency.

In this paper, each event log is represented by a set of event codes in sequential order. More details will be explained in chapter 3.2.

2.3.3. CATEGORICAL DATA ENCODER

Since each event code is represented by a string, it needs to be encoded into a numerical data type that the model can understand.

ONE-HOT ENCODEING

One hot encoding is often used to encode categorical data. It creates one vector in length K (the number of unique event types) for each event type to against other event types. In the vector of each event code, the ith is 1 if the event code is in ith event type and the remaining positions are 0. A sparse vector can be created for all possible event types. However, if they are many event types in the data, the dimension of the feature vector

can become very high. Also, since the presence of each event code is hard encoded by 1 or 0, the distance between any two events' vectors is $\sqrt{2}$ that the feature vector does not contain the relations between event codes.

2

WORD EMBEDDING

Word2vec [Goldberg and Levy(2014)] is one of the most popular word embedding models in the Natural Language Processing area. It was designed to vectorize the huge number of vocabularies in a low dimensional space. It assumes that there are related words in the corpus and these related words are likely to present in a similar context. The core idea of word2vec model is to map the input vector from K (the size of the vocabulary) dimension to dw (the self-defined embedding dimension) and the learned vector representation is good at predicting the nearby words. The similarity of two words can be represented by the cosine distance between their corresponding vector representation. A famous pre-trained word2vec model provided by Google [Mikolov(2013)] was trained on 100 billion words from a Google New dataset and successfully mapped a vocabulary of 3 million words to a set of feature vectors in length 300.

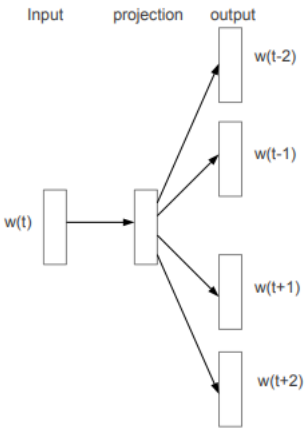


Figure 2.4: The model structure of word2vec. The training objective is to learn word vector representations that are good at predicting the nearby words. [Mikolov *et al.*(2013)Mikolov, Sutskever, Chen, Corrado, and Dean]

There are several parameters in the word2vec model.'

- Size: The embedding dimension. A small dimension can improve the efficiency of the model. But if the dimension is too small, detailed information may be lost and many vectors may be overlapped in the embedding space.

- Window: Maximum distance between the current and predicted word within a sequence. Related events are usually adjacent but not always. Assigning an appropriate window size help to explore the related events in some distance. The window size here is different from the window size used while windowing. We use the default value 5.

- Min-count: Ignore all words with a total frequency lower than this. Since one of the goals of this paper is to detect anomalies and anomalies are usually rare, we keep min-count equals 1 in this paper which means we are going to encode all the events presented.

2.3.4. RECURRENT NEURAL NETWORK

One of the most classic and popular deep learning algorithms for sequence modeling is the recurrent neural network. The recurrent neural network is naturally suitable to model sequences due to its directed graph structure. In recent years, there are some advanced recurrent neural networks proposed to overcome the drawbacks of the basic recurrent network and improve the performance.

RECURRENT NEURAL NETWORK(RNN)

Recurrent Neural Network (RNN) was proposed by [Rumelhart *et al.*(1988)Rumelhart, Hinton, Williams *et a*
The structure of RNN is shown in 2.5. Connections between RNN units form a directed graph along the input sequence. RNN uses the hidden states of units to memorize the input information and deliver the memory to the next unit until reaching the end of the sequence.

The key idea of RNN is to map the input vectors to a hidden dimension space. Given a sequence in the form of $x = (x_1, x_2, \ldots, x_T)$ as the input to LSTM, at each time step, it feeds one element in the input to an RNN unit and feed-forwards the current hidden state to the next unit until the end. A dense layer is appended to map the last hidden state back to the dimensional space of the output. This structure makes RNN is naturally suitable for temporal sequence modeling tasks such as speech signal recognition and natural language processing. However, when the sequence is very long, the memory of the elements at the beginning can be lost that the gradient of earlier units vanish exponentially until the model can not learn. It is the so-called gradient vanishing[Hochreiter(1998)].

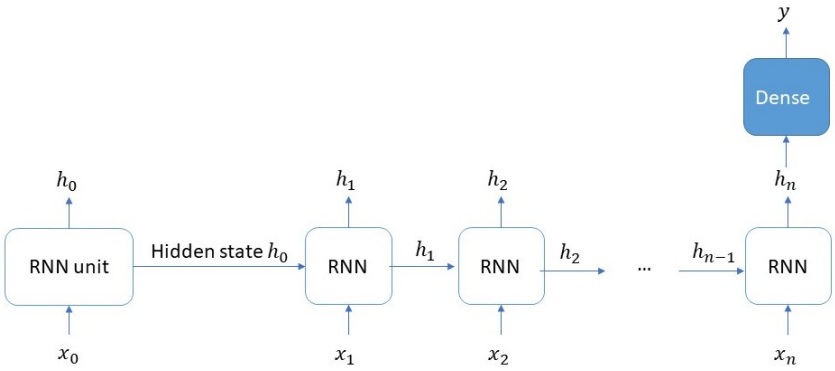


Figure 2.5: Recurrent Neural Network

2

LONG SHORT-TERM MEMORY NEURAL NETWORK (LSTM)

Long short-term memory (LSTM) [Hochreiter and Schmidhuber(1997)]is an improved algorithm of traditional RNN that overcomes the vanishing gradient issue by choosing the memory to memorize or forget. Instead of passing all memory in units like RNN, LSTM uses three gates to control the memory flow (Figure 2.6): forget gate, update gate and output date. These gates selectively decide the memory to memorize or forget to avoid memory exposure.

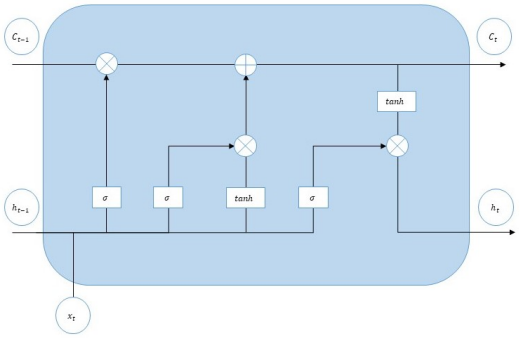


Figure 2.6: Long-short term memory unit [Olah(2015)]

- Forget gate
  The first gate is the forget gate which is used to decide what information it is going to abandon from the cell gate. The gate is made by a sigmoid layer. The decision is based on the hidden state at last time step h_{t-1} and the input at the current time x_t multiplied by corresponding parameters respectively. When $W_f = 1$, $U_f = 0$, it represents that all previously memorized information will be thrown away from the cell.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.1}$$

- Input gate
  The input gate layer uses a sigmoid layer to decide with values to update.

$$i_t = \sigma(W_i x_t + U_i h_{ht-1} + b_i) \tag{2.2}$$

  A tanh layer creates the candidate values that could be added to the cell state.

$$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{2.3}$$

  The next step is to multiply the old cell state Ct-1 with forget gate and multiply the candidate update with the input gate to update the forgetting old cell state with the new one.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.4}$$

- Output gate
  Not everything the model memorized is always needed to output. The output gates decide what parts of the cell state we are going to output and what other parts of the cell state we just store them in the chain for future use.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.5}$$

  We put the cell state through a thah layer and multiply the result of output gate and only output the parts we want. The output hidden state vector is:

$$h_t = o_t * tanh(C_t) \tag{2.6}$$

[Du *et al.*(2017)Du, Li, Zheng, and Srikumar] was the first paper that introduced LSTM to software event log analysis and anomaly detection. The LSTM model was trained on a large number of normal event log sequences to learn the dependencies among events in the event logs and predict the next events after the input sequences. Given a new event log, events which deviated from the model were likely to be anomalies. Using LSTM, the author managed to acquire a 99.994% anomaly detection accuracy on the HDFS log dataset [Xu *et al.*(2009)Xu, Huang, Fox, Patterson, and Jordan]. In this paper, I will use the work from this paper as the baseline and aim to improve the performance and interpretability of the model.

3

Data Exploration

In this chapter, I explain how the training, validation and testing dataset were collected, separated and parsed. Since each event log in the dataset is a plain text file, I firstly parse the document into a sequence which contains a set of event and convert each long sequence into a set of short sequences using a sliding window. During the experiments, I observed many fake anomalies and analyzed the reason why these events were incorrectly as fake anomalies. After consulting the expert in ASML and the current ASML diagnostic tool document, several filters are applied to remove these observed noises. Once finishing the above data processing, each log file was processed to a set of short sequences and each sequence contains the fixed number of event codes. This kind of data is ready to feed the model proposed in the next chapter.

The entire data pre-processing flow was shown in Figure 3.1

Figure 3.1: Data pre-processing

3.1. DATA COLLECTION

TestArena is a platform where test engineers can submit and execute tests. The event log can be generated during the execution of the test. We can request the event logs of a specific test by searching its test ID. The data I used is a collection of logs recording the activities of a test task which is the largest in the current ASML testing platform.

The dataset includes 497 event log files downloaded from TestArena. The event logs are firstly separated into two classes by their final status (successful/failed). The successful event logs are then split to 90% and 10% as the training set and validation set. And I will diagnose the failures on the failed event logs.

3.2. LOG PARSING AND EVENT REPRESENTATION

Each event log in the dataset is an unstructured plain text file that contains many text blocks and each block records the information of one event. An example is shown in Figure 3.2. First of all, I separate the text file into blocks by empty lines and split each text block into words by punctuations such as space, bracket, and comma. Each block is an event message with several attributes. These attributes have been introduced in the chapter 1.2.

In many related works, the timestamp and the event format are two attributes usually selected to represent the event as mentioned in chapter 2.3.2. In this paper, I only use the event code for the following reasons.

Using the event code to represent the event is enough because:

- In the ASML event log, each event code corresponds to a distinct event format. For instance, as shown in the Figure 3.2, event CO-0417 corresponds to an event format - *String too long. xxx is empty or exceeds 75 characters.* and event CO-0428 corresponds to another event format - *Performance indicator specification holds error, item not included. xxx not included.* By searching the distinct event code, we could acquire all the related useful information of the event such as the event status, text description and the component it belongs to. Therefore, using the event code is enough for us to judge whether it is abnormal.

The timestamps of events are usually used to monitor the time-difference among events. There are two reasons why the timestamp attribute was discarded.

- The events are logged at the time when they are executed rather than in a fixed interval. And the runtime of each event is usually not static but more or less varies. It is normal if the runtime varies in a safe range. Table 3.1 demonstrates how the duration of different tests varies.

- If the runtime is unusually long, there are a set of events as shown in Table 3.2 reporting the time-out activity. So the runtime status can be reflected by the occurrence of this kind of alarm events.

Therefore, in this ASML data, the exact timestamp or time interval between two events is not very useful. The sequential order is much more important than the chronology. For example, one event unexpectedly occurs before another could lead to a fault.

File	Duration
1	2 hours 59 minutes 37 seconds
2	2 hours 47 minutes 35 seconds
3	3 hours 12 minutes 24 seconds

Table 3.1: Varied duration of different tests

Event code	Description
CN-0001	Timeout occurred.
	Timeout occurred for function/event xxx and to server xxx
CN-00C0	Timeout in message reception.
	Closed connection due to read timeout of 3630 seconds.
CN-00C3	Timeout in message reception.
	Timeout on reading message; read: 0, toread: 24.
CN-00C4	Client closed the connection.
	Client has closed connection with server

Table 3.2: Timeout events

Overall, each log file can be parsed into a sequence that contains a set of categorical event codes in the sequential order.

3.3. DATA FILTER

Due to the complexity of the ASML machine and the delay of logging, the log sequence is very noisy that can in an unnecessary long length. The noise can hide the real patterns in the data and make the training less efficient. During the experiments, I observed many fake anomalies and most of these events were incorrectly detected as fake anomalies due to the following reasons:

- They do not have causality with their nearby events.
- They are the duplicates events of prior events.
- They are in the execution loops.

After consulting the diagnostic engineers and checking the ASML diagnostic tool documents, several filters are applied to specifically remove these known noisy events. Filtering these noise can help reduce the false positives and shorten the length of event log sequence.

3.3.1. FILTERING DEACTIVATED STATUS EVENTS

DEACTIVATED event (Figure 3.2) occurs when the system attempts to recover from the previous error. The presence of this event indicates a recovered error and is irrelevant to the previous sequential relations. Therefore, all DEACTIVATED events with the event code ER-0FFF were removed from the data.

```
05/10/2019  18:10:06.0624  Machine:GW02  (Rel:9.9.9.d, CODS [17761], CODS_list.c, ?.?, 595)
SYSTEM ERROR: CO-0417 DEFAULT
String too long
Internal Name () is empty or exceeds 75 characters.

05/10/2019  18:10:06.0627  Machine:GW02  (Rel:9.9.9.d, CODS [17761], CODS_list.c, ?.?, 949)
SYSTEM ERROR: CO-0428 DEFAULT (linked to CO-0417 CO-0417)
Performance indicator specification holds errors, item not included.
PI not included; dd: MExPI:param_list_type, groupid: ME, ext.name: , int.name: , storename: FastTools, type name: double

05/10/2019  18:10:06.0628  Machine:GW02  (Rel:9.9.9.d, CODS [17761], CODS_list.c, ?.?, 950)
SYSTEM EVENT: ER-0FFF DEFAULT (linked to CO-0428)
DEACTIVATE: CO-0428
DEACTIVATE: CO-0428
```

Figure 3.2: Deactived status event. Event ER-OFFF deaciviated the prior error CO-0428.

3.3.2. FILTERING EVENTS IN THE BLACKLIST

There are some events verified to be useless for analyzing the test. For example, some DN-XXXX events occur at the time when the logging component logs one event, so they can be everywhere and have no causality with adjacent events; some TH-XXXX events initialize components and only occur at the beginning of the test; Some ER-XXXX events record the status of the log file which is not interesting to know. These events are irrelevant to the status of the test but are easily reported as false positives by a sequential anomaly detection model due to their low connections with adjacent events. After consulting the expert, a blacklist as shown in Table **??** is built to filter these known irrelevant events.

Event code	Description
FT-0114	one or more tables are not closed yet. FTDS closed some databases that were left open.
DN-0603/DN-0604/DN-0605	Log xxx Event
ER-4101	ER logging task initialized
ER-4102	Log file xxx is created. Log file xxx is closed.
ER-4103	Log file xxx is closed. Log file xxx is created.
PY-1000	Command executed outside SM context
TH-1112	xxx is double defined, exists as a group member
TH-0036/TH-0037 /TH-0038/TH-003B	THCC sim component initialization.
TH-004D	Group has an instance named like itself.
TH-1000	Non-default component simulation modes.
TH-2300	Sum of declared diagnostic trace buffers is more than allowed.
TH-2100	Sum of declared realtime trace buffers is more than allowed.

Table 3.3: Black list of Irrelevant events

3.3.3. FILTERING DUPLICATES

Sometimes, there are a couple of events having the same event codes but different parameters. For example, in the following fragment of the log displayed in the figure 3.3, event coder TH-1112 represents an event template "*Parse warning in the config file. Component 'XXX' is double defined, exists as a group member.*" with a parameter – component name XXX. Since I only use the event code to represent the event, this kind of events become indistinguishable after abandoning the corresponding parameters. The above fragment was parsed to a list: [TH-1112, TH-1112, TH-1112, ..., TH-1112]. In the current ASML diagnostic tool document, it assumes that whether a given event was logged once or more times is not relevant. Therefore, I only keep the first event and abandon the remaining duplicates.


```
05/10/2019  23:59:05.7900  Machine:IP83  (Rel:9.9.9.d, THCM [17193], THCCxTH_errors.c, ?.?, 2759)
SYSTEM WARNING: TH-1112 DEFAULT
Parse warning in config file
Component 'ME' is double defined, exists as a group member

05/10/2019  23:59:05.7901  Machine:IP83  (Rel:9.9.9.d, THCM [17193], THCCxTH_errors.c, ?.?, 2759)
SYSTEM WARNING: TH-1112 DEFAULT
Parse warning in config file
Component 'RS' is double defined, exists as a group member

05/10/2019  23:59:05.7903  Machine:IP83  (Rel:9.9.9.d, THCM [17193], THCCxTH_errors.c, ?.?, 2759)
SYSTEM WARNING: TH-1112 DEFAULT
Parse warning in config file
Child 'RS' is double defined, exists as a group member

05/10/2019  23:59:05.7905  Machine:IP83  (Rel:9.9.9.d, THCM [17193], THCCxTH_errors.c, ?.?, 2759)
SYSTEM WARNING: TH-1112 DEFAULT
Parse warning in config file
Child 'RQBG' is double defined, exists as a group member

05/10/2019  23:59:05.7906  Machine:IP83  (Rel:9.9.9.d, THCM [17193], THCCxTH_errors.c, ?.?, 2759)
SYSTEM WARNING: TH-1112 DEFAULT
Parse warning in config file
Child 'RQBV' is double defined, exists as a group member
```

Figure 3.3: Duplicate event TH-1112

3.3.4. FILTERING LOOPS

The loop in the event log sequences is defined as a set of events that are repeatedly executed for several times. An example is given in Figure 3.4. Event SM-9000 reports the initial request and event SM-9402 reports successful initialization status. When there are many components in the test need to be initialized, these two events always occur one after another one until finishing all the initialization. Loops are every common in sequence. A loop reflects a small functional unit. Loops sometimes can have a negative influence on the prediction of the model. Imagine we have a sequence with length 10 and a pair of SM-9000 and SM-9402 recurrently occurs for 5 times:

$$SM-9000SM-9402\cdots SM-9000SM-9402$$

In principle, after finishing all the initialization, the test will come to the next testing phase. For example, the test could start configuring: *Event WT-2001: WTC hardware configuration detection.* However, in practice, I found that the casual dependency between SM-9000 and SM-9402 was so strong that whenever an event SM-9000 occur, the model was most likely to predict the next event as SM-9402. In this case, event WT-2001 would be wrongly detected as an anomaly. Therefore, a filter is applied to filter the extra

```
08/16/2019  11:17:14.3100  Machine:GW02  (Rel:9.9.9.d, SMDC [23068], SMDC_init_call.c, ?.?, 1046)
SYSTEM EVENT: SM-9400 DEFAULT
Initialize request for driver ZDFR.
Initialize request for driver ZDFR .

08/16/2019  11:17:14.3106  Machine:GW02  (Rel:9.9.9.d, SMDC [23068], SMDC_init_call.c, ?.?, 332)
SYSTEM EVENT: SM-9402 DEFAULT
The CTRW1 driver has been successfully initialized.
Driver CTRW1 [phase: Base] has been successfully initialized.

08/16/2019  11:17:14.3112  Machine:GW02  (Rel:9.9.9.d, SMDC [23068], SMDC_init_call.c, ?.?, 1046)
SYSTEM EVENT: SM-9400 DEFAULT
Initialize request for driver CTRW1.
Initialize request for driver CTRW1 [phase: Full] .

08/16/2019  11:17:14.3118  Machine:GW02  (Rel:9.9.9.d, SMDC [23068], SMDC_init_call.c, ?.?, 332)
SYSTEM EVENT: SM-9402 DEFAULT
The CTRW2 driver has been successfully initialized.
Driver CTRW2 [phase: Base] has been successfully initialized.

08/16/2019  11:17:14.3124  Machine:GW02  (Rel:9.9.9.d, SMDC [23068], SMDC_init_call.c, ?.?, 1046)
SYSTEM EVENT: SM-9400 DEFAULT
Initialize request for driver CTRW2.
Initialize request for driver CTRW2 [phase: Full] .

08/16/2019  11:17:14.3131  Machine:GW02  (Rel:9.9.9.d, SMDC [23068], SMDC_init_call.c, ?.?, 390)
SYSTEM EVENT: SM-9402 DEFAULT
The ZDFR driver has been successfully initialized.
Driver ZDFR has been successfully initialized.
```

Figure 3.4: An example of the loop in the sequences

events in the loops. As the lengths of all loops are unknown, I specifically filter the loops in length of two and three which are the lengths of the most frequent loops.

3.4. NUMERICAL REPRESENTATION

The categorical event codes need to be converted to numbers that the model can understand. We simply build a string-to-integral look-up table where each categorical event code corresponds to a unique number. The size of the dictionary equals to the number of event code types in the entire dataset.

3.5. WINDOWING

Since each event log usually contains thousands of events, it is impossible to build one long vector for the entire sequence. Therefore, a sliding window method is applied to separate the long sequence into a set of shorter fixed-length sub-sequences. Firstly, a sub-sequence is sliced by a window in T length, and at each time, the window shifts forward one step. The overlapping windows could avoid the loss of information of events on the boundaries of the windows. Then, all chucked sequences are appended together to build a large 2-dimensional array as the input of the model.

3.6. SUMMARY

The raw dataset is a collection of plain text log files. The log files of successful tests are used to train and validate and the log files of failed tests are used to detect and diagnose anomalies. Several filters are applied to remove noisy events in the data. These filters are proposed by myself after observing the patterns of false positives. After that, long log sequences are separated into many short sub-sequences by a sliding window in length T. Consequently, we can have a set of sub-sequences where each sub-sequence has T event codes.

4

MODEL

In this chapter, I proposed an attentional long-short term memory model to detect and diagnose anomalies. Recall the characteristics of our data and the goal of this paper, there are some important notes when selecting the algorithms.

- Each event log of a test is labeled by the status of the test - successful or failed. However, the annotations of events are not available. This matters when choosing the supervised or unsupervised algorithm.
- The events in the event logs are chronologically logged and assumed to be causally related. Due to the non-synchronous of the logging system and the complicated parallel processes in the test, the distance between two related events is not static. The algorithm should be able to robustly model the causal dependencies in any distance.
- The result of the algorithm should be interpretable to help people understand the analysis process behinds the predictions given by the model.

For the first note, the way to process and model the sequence was inspired by Deeplog [Du *et al.*(2017)Du, Li, Zheng, and Srikumar]. In this paper, given a sequence in length T, the author used the first T-1 events as the input and use the Tth event as the label to predict. Each event type was a class. The sequence modeling problem was therefore converted to a K-classification problem where K is the number of event types. In the paper, the author used an LSTM model to model the sequence. After training, given a new sequence, the model can predict the next event under the premise of its prior events. If the actual next event is different from the prediction, this event then is likely to be an anomaly.

However, in the LSTM, even though there is not a score of each element in the input specified to measure its importance to the prediction, the last event in the input actually has the most effect when predicting the following event. This kind of effect can be understood by analogy with the joint probability that the last multiplied element has the most effect on the final multiplication result. In order to avoid this bias, I introduce an

attention mechanism [Zhou *et al.*(2016)Zhou, Shi, Tian, Qi, Li, Hao, and Xu] to measure the importance scores of elements in the input. This attention mechanism was proposed and applied to solve attentional text classification tasks. The outputs of LSTM are weighted by their corresponding attention weights. The attention weights are optimized to give better prediction and the element with the highest weight indicates a relatively stronger relation to the predicted event. Therefore, we can understand the prediction by observing the attention weights in the model. This algorithm is so-called attentional long-short term memory model. As far as I know, this paper is the first one applying the attentional LSTM prediction model on anomaly detection and diagnosis task.

4.1. MODEL STRUCTURE

After data pre-processing, we have a set of sub-sequences where each sub-sequences has T event codes. The ith sub-sequence is in the form:

$$x_i = x_{i0}, x_{i1}, x_{i2}, \ldots, x_{iT-1}, x_{iT}$$

I use the first $T-1$ events $(x_{i0}, x_{i1}, x_{i2}, \ldots, x_{iT-1}$ as the input, and the last event (x_{iT}) as the label to predict. Since there are 664 types of events in the dataset, we have 664 classes in total.

The model was designed in Keras and had five layers 4.1:

1) The input layer receives the pre-processed data.

2) The Embedding layer converts the input into a 3-dimensional tensor and reduces the dimension.

3) The LSTM layer learns the sequential dependencies of the input by choosing the information to forget or memorize.

4) The attention layer learns the attention distribution with respect to inputs when the model is making decisions.

5) The SoftMax layer calculates the likelihoods of all possible predictions.

The model was built to model the dependencies of the input sequence and output the likelihoods of all classes under the premise of the input sequence. During the inference phase, the model also outputs the attention scores of elements in the input. The entire pipeline is shown in Figure 4.1.

4.2. VECTOR REPRESENTATION

One-hot encoding is one of the most common approach to build the vector representation for categorical data. If we use one-hot encoding, as there are 664 types of events in the dataset, each event will be represented by a sparse binary vector in the length of 664. Using this kind of high dimensional sparse representation vectors can cause the training of the model less efficient. Therefore, a word2vec model is applied in the embedding layer to represent events with lower dimensional vectors. The word2vec is trained on the entire dataset including normal sequences and abnormal sequences. The training

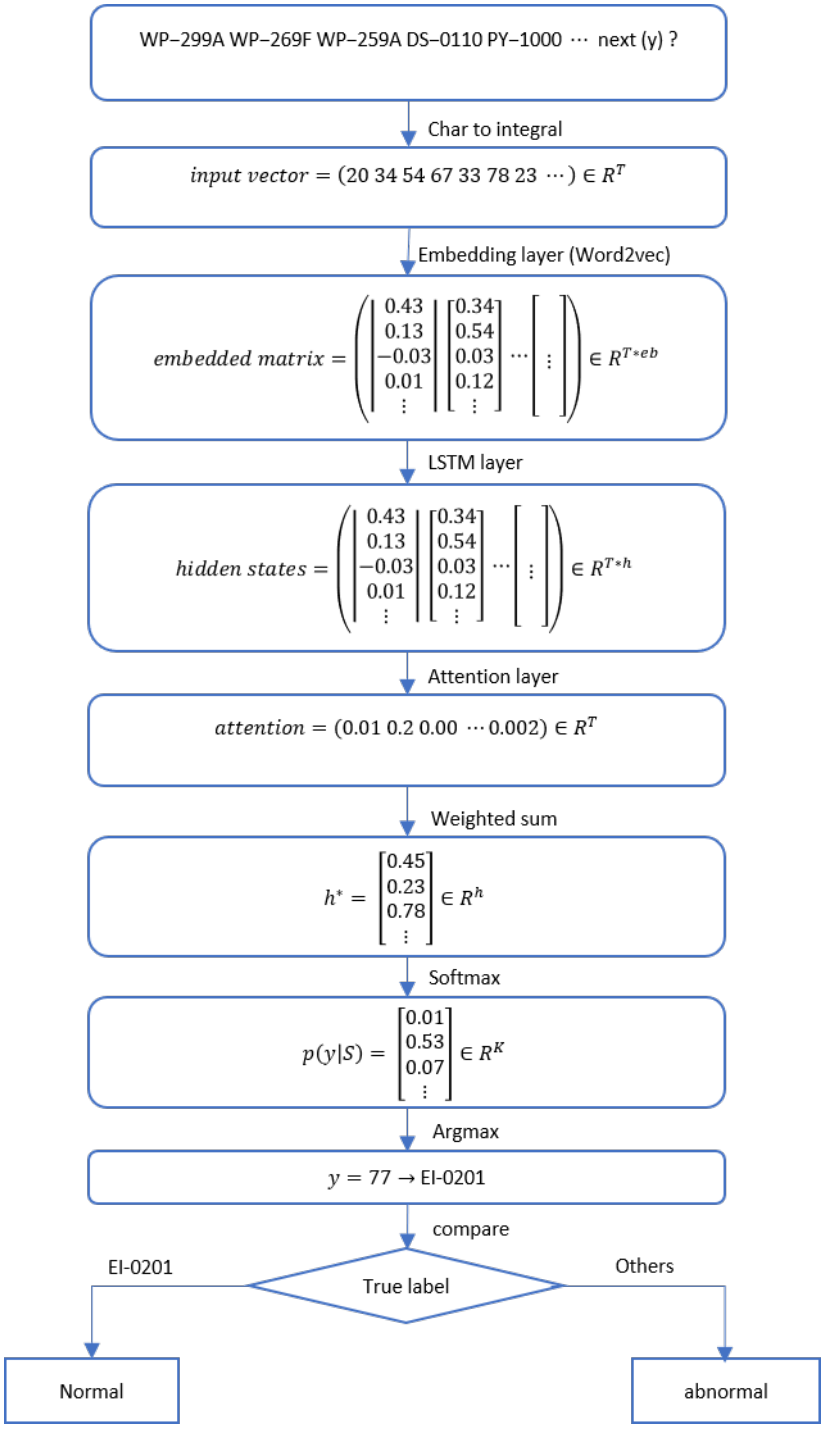
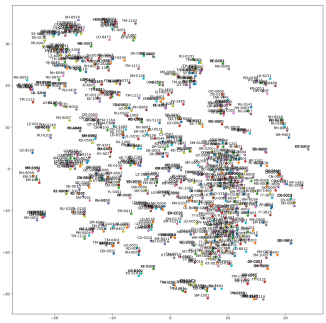


Figure 4.1: Model structure

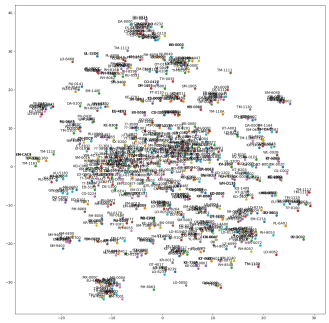
4

objective is to learn the vector representation of each event that can predict its nearby events. Events that always occur in the same context will have similar vector representations.

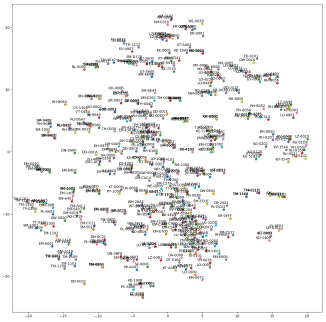
Using lower dimensional vectors to represent events can improve the efficiency of the training. But for a large event dictionary, using very low dimensional vectors might cause the loss of distinct information of events during the dimension compression. To intuitively understand the influence of the size of embedding dimension, I visualize different dimensional vector representations of all events as shown in Figure 4.2. Since it is impossible to visualize the data in high dimensional space, vectors are compressed into 2D using T-distributed stochastic neighbor embedding (T-SNE). Note T-SNE is only used for visualization purpose and not relevant to the proposed approach of this paper. Other dimension reduction approaches might also work.



(a) Embedding dimension 16



(b) Embedding dimension 32



(c) Embedding dimension 64

Figure 4.2: A 2D visualization of different dimensional vector representations. Note there might be information loss during the dimension compression, but the points which are close in 2D space will also have short distance in the original dimensional space.

In terms of the three graphs in Figure 4.2, we can see the points on the graph are more dense when choosing smaller embedding dimension that some points in the graph 4.2(a) are overlapped. That means choosing the lower embedding dimension might take the risk of losing distinct information of some events even though the training is faster. The performance of using different embedding dimension will be compared in chapter 5.

In Figure 4.2, the distance between any two points in the graph reflects the similarity of their corresponding vectors. Related events that frequently occur in the same context are likely to have similar vector representations and vice versa. This similarity of events can be explained by an example as shown in 4.3. There are two points in the middle of the graph that very close: TM-1162 and TM-1164. In terms of their text descriptions from the event log as shown in Table 4.1, they both relate *Remote Test Manager Service* and always occur one after another, so they have similar vectors representations and therefore are visualized by two close points in the graph.

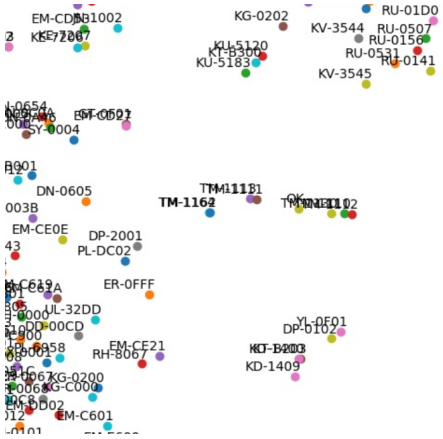


Figure 4.3: An example of explaining how the relative distance between two points in the 2D visualization reflect the similarity of their corresponding events.

Event code	Log description
TM-1162	Remote Test Manager Service is disabled.
TM-1164	Remote Test Manager Service: CPD finished.

Table 4.1: The log description of two overlapping events.

There are some advantages of using word2vec compared to the one-hot encoding. One of the main advantages is that the dimension of the vector representation is reduced from K to dw(embedding dimension). The mapping is very useful especially when there are many event types in a large dataset.

When using one-hot encoding, each event is represented by a vector which includes a one and many zeros. For any two events, the cosine similarity between their vectors is always zero. Compared to one-hot encoding which ignores the underlying relations

among events, word2vec analyzes the context of events when learning their representations. The relation between any two events can be reflected by the cosine similarity between their associated vectors. More similar vectors indicate that these two events are more related. Sometimes, the relative order between two concurrent events is random due to the asynchronism of logging system. Using context-based representation can avoid the model getting confused for the unpredictable relative order between events in this situation. In another case, some events from different processes that have the same functionality might were named as different event codes by different people at different time.

When two events always concurrent together but the relative order between them is arbitrary, the model will consider these two events are related and assign two similar vectors to them so that the model would not get confused for the relative order of these two events. Another case is that the events that have similar functionality but in different event codes can have similar vectors as well.

4.3. LONG-SHORT TERM MEMORY

Recalling chapter 2.3.4 that Long short-term memory (LSTM) is an improved algorithm of traditional Recurrent Neural Network (RNN) that overcomes the vanishing gradient issue. In recent years, it has shown impressive performance on many sequence modeling tasks. The general workflow of LSTM is shown in Figure 4.4.

Give the input of sequence in the form of $x = (x_1, x_2, \ldots, x_{T-1})$ as the input to LSTM, it feeds one event to an LSTM unit at each time step and maps the embedded event vector to a hidden state vector. It passes the hidden state to the next unit until reaching the end of the input sequence. The hidden state at each time step depends on the current event input and the hidden state received from the last unit which contains the memory of all prior events.

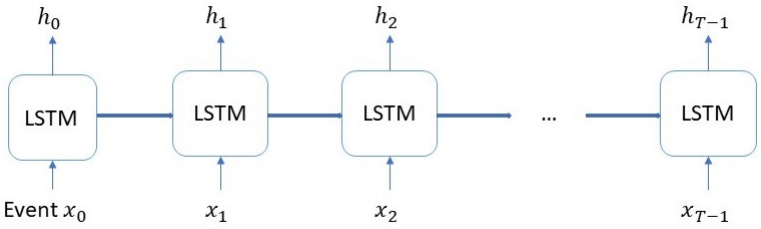


Figure 4.4: Long-short term memoy

The output of the LSTM layer is a set of hidden states at each time step in the form:

$$H = (h_1, h_2, \cdots, h_{T-1})$$

where $h \in \mathbb{R}^h, H \in \mathbb{R}^{T \times h}$. T is the length of the sliding window and h is the hidden dimension of the LSTM unit.

The basic LSTM units learn the elements in the inputs and deliver the states forward from beginning to the end. The bidirectional LSTM [Graves and Schmidhuber(2005)] appends a second backward layer of LSTM units and learns the elements from the end of the sequence to the beginning. The final hidden states of at each time step are the concat result of forwarding hidden state $\overrightarrow{h_t}$ and the backwarding hidden state $\overleftarrow{h_t}$ as shown in Figure 4.5. The bidirectional LSTM outperforms the single directional LSTM when an event in the future can indicate the occurrence of the events in the past so that the state of one unit can rely on both its before and after states. In chapter 5, I will compare the performance of LSTM and bidirectional LSTM.

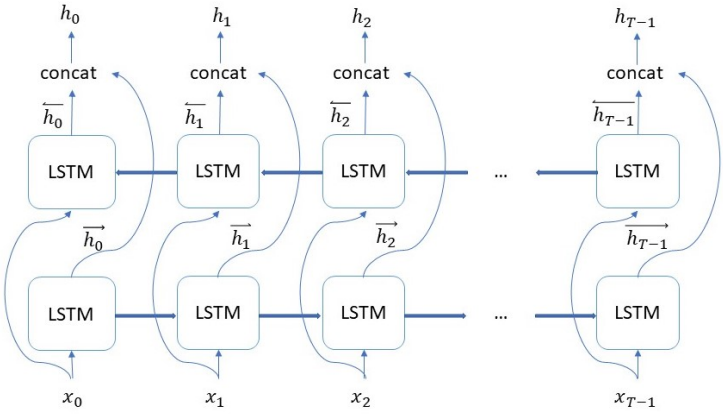


Figure 4.5: Bidirectional LSTM

4.4. ATTENTION MECHANISM

In general, as described in chapter 2.3.4, the LSTM takes the hidden state of the last unit to represent the entire input sequence and uses it to predict the output. However, we can not sure the prediction is really under the premise of the entire input sequence or just based on the last event of the sequence. Therefore, an attention mechanism is proposed to open the black box of LSTM. The idea of using attention mechanism came from a text classification paper [Zhou *et al.*(2016)Zhou, Shi, Tian, Qi, Li, Hao, and Xu] in Natural Language Processing. In this paper, the author used the the attention mechanism to learn which words in the document were more important to the document classification.

Instead of directly using the last hidden state of LSTM to predict, an attention layer is applied to the LSTM outputs to measure which events in the input sequence are more important for predicting the next event. The key idea behind of attention mechanism is to learn a function which maps a hidden state vector h_i to a weight score e_i as shown in 4.1. e_i is used to measure the importance of each event in the input on the prediction of the label.

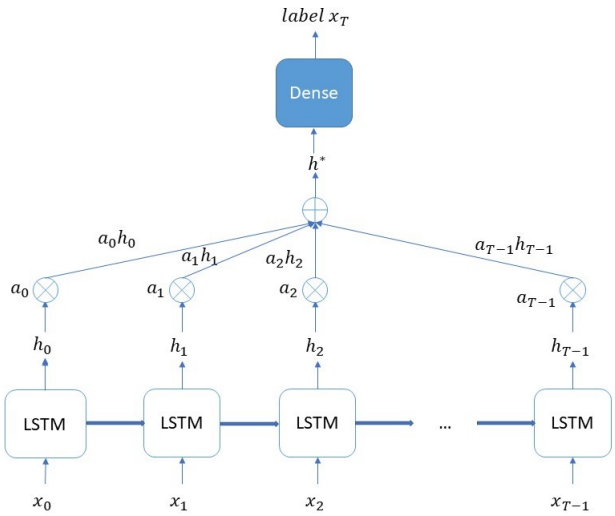


Figure 4.6: Attention layer

$$e_t = V_\alpha^T tanh(U_\alpha H + b_\alpha) \tag{4.1}$$

A SoftMax function 4.2 is applied to normalize the scores in the range of [0,1].

$$\alpha = \frac{exp(e_t)}{\sum_{t=1}^{T} e_t} \tag{4.2}$$

The application of the attention scores is computed as 4.3. The representation $h^\star$ of the input sequence is a weighted sum of the hidden states of LSTM.

$$h^\star = H\alpha^T \tag{4.3}$$

Note $V_\alpha \in \mathbb{R}^h, U_\alpha \in \mathbb{R}^{h\times h}, e \in \mathbb{R}^T, \alpha \in \mathbb{R}^T, h^\star \in \mathbb{R}^h$.

4.5. SOFTMAX ACTIVATION FUNCTION

The Softmax activation function is commonly used in the multi-class classification problems. A dense layer with softmax activation function is applied to map the weighted hidden state of attention layer to the dimension space of outputs.

$$p(Y|S) = softmax(Wh^\star + b) \tag{4.4}$$

In the equation 4.4, W is a matrix of dimension $K \times H$ where K is the number of event types and H is the hidden state dimension of LSTM. The output of SoftMax layer is a probabilistic vector in in the length of K of predicting $Y : (x_T = K_0, x_T = k_1, \ldots, x_T = x_K)$ under the premise of input sequence $S : (x_0, x_1, \ldots, x_{T-1})$. $p(x_T = k_i|s)$ represents the likelihood of predicting events x_T to be k_i.

We can then use an argmax function (Equation 4.5)to acquire the prediction of x_T with the highest probability.

$$x_T = \arg\max_Y P(Y \mid S) \tag{4.5}$$

4.6. LOSS FUNCTION

Categorical cross entropy is a typical loss function used in the multi-class classification problem when the last layer of the model is a softmax activation function. The equation of categorical cross entropy is defined in Equation 4.6.

$$loss = -\sum_{k_i \in K} y(k_i) \log(k_i) \tag{4.6}$$

where k is the predicted class (event code) and $y(k)$ is its corresponding likelihood. Compared to other classic loss function such as Mean Squared Error (MSE), CE can not only reflect the classification accuracy but also the confidence of the classification.

Using categorical cross entropy loss requires the target is encoded by a one-hot encoder. In this case, only the event class present is encoded by 1 and the remaining are 0, the loss function can be simplified to Equation 4.7.

$$loss = -y(k_i) \log(k_i) \tag{4.7}$$

4.7. ANOMALY SCORING

Once the model is able to understand the characteristics of normal sequences, it can detect anomalies by identifying misclassification with high anomaly scores. High anomaly scores indicate that the event does not conform to the expected behavior of the training model and is therefore unlikely to be predicted as the next event. Therefore, events with relatively high anomaly scores are more likely to be anomalies.

The anomaly score is defined by the relative loss. Terms for relative loss are introduced as below:

- Relative loss (RL) [Bontemps *et al.*(2016)Bontemps, McDermott, Le-Khac *et al.*]
  The relative loss measures the deviation between the actual value and the model by calculating the categorical loss difference between the actual event and the predicted event as shown in Equation 4.8. The prediction probability given by the model reflects the calibration of the model. A higher probability means the model has more confidence in its predictions. Figure 4.7 shows that, in most cases, our model can give predictions with high reliability. Note for the event correctly predicted by the model, the relative loss is zero.

$$RL = loss(y) - loss(\hat{y}) \tag{4.8}$$

- Anomaly score threshold
  The anomaly score threshold is determined by the normal sequences from the validation dataset and the abnormal dataset from the testing dataset. Figure 4.8

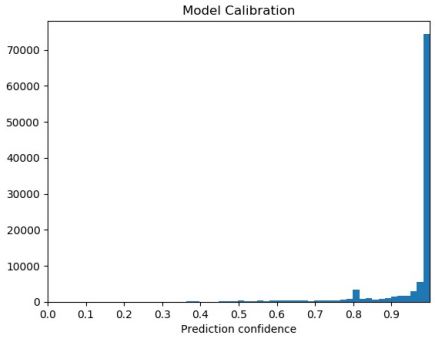
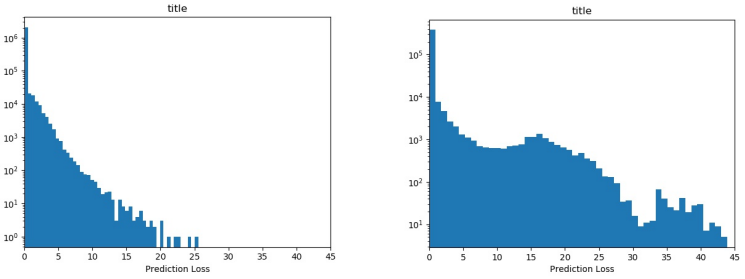


Figure 4.7: Model calibration. It is reflected by the highest probability of the output given by the model at each time step.

shows the dense distributions of anomaly scores in normal sequences and abnormal sequences. (Note the y-axis in the plot is non-uniform.) As can be seen from the left figure, the anomaly scores of most events in the normal sequences are low and the dense of higher anomaly scores decreases steadily. Conversely, there are more events associated with higher anomaly scores in abnormal sequences as shown in the right graph. These two different dense distributions of anomaly scores indicate that we can choose the threshold of anomaly score to distinguish the true anomalies detected in abnormal sequences from the benign anomalies detected in the normal sequences.

The exact threshold value is determined using the confusion matrix.



(a) Normal sequences using log-yaxis (b) Abnormal sequences using log-yaxis

Figure 4.8: Dense distribution of anomaly scores in normal sequences from validation dataset(left) and abnormal sequences form testing dataset(right). Note the y-axis in each plot is non-uniform.

- Confusion matrix
  The confusion matrix can be used to measure the performance of anomaly detection when using different thresholds. The definitions of terms in the confusion matrix are shown in Table 4.2.

True positive (TP)	The anomalies detected from abnormal sequences. These anomalies may relate to the root cause of failed tests.
False negative (FN)	The normal events in the abnormal sequences.
True negatives (TN)	The normal events in the normal sequences.
False positive (FN)	The benign anomalies detected from normal sequences. In principle, there should not by any anomalies in the normal sequences. One of the reasons for these false positives could because of the imperfect classification capability of the model. Another main reason could be that the behavior of these anomalies indeed occur in the normal sequence but not frequently enough that the model can learn.

Table 4.2: The definition of terms in the confusion matrix

We specifically use the precision and false positive rate to evaluate the performance of thresholds. The precision evaluates the number of anomalies from abnormal sequences out of all anomalies detected by the model. And the false positive rate evaluates that in normal sequences, how many events are mistakenly reported as anomalies. We want to determine the anomaly score threshold by asking the model to distinguish the anomalies in abnormal sequences from the fake anomalies in normal sequences.

$$Precision = \frac{TP}{TP + FP} \tag{4.9}$$

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{N} \tag{4.10}$$

4.8. UNDERSTANDING THE ANALYSIS PROCESS OF THE MODEL

Recalling Section 4.4 where we introduce the attention mechanism, a set of attention scores that indicate the importance of input elements are associated with input sequences. By visualizing the attention scores distribution, we could easily find out the most important prior event which contributes the most to the prediction of the query event. Therefore, high attention scores indicate a strong relationship between the prior event and the query event.

We could use the same way to understand anomalies. Recall the definition of the anomaly is the event that does not conform to the model which has been trained on a large number of normal sequences. By exploring the related prior events of an anomaly, we could know the reason why the model gives this prediction that leads this event becomes an anomaly.

5

RESULTS

In this chapter, I compare the training performance of our models with Deeplog in different parameters. The model with the best training results is used to detect anomalies in the testing dataset. And the anomaly score threshold is decided by minimizing the number of anomalies in the normal sequences. Moreover, a visual interface is built to help users intuitively localize and analyze the anomalies.

5.1. TRAINING

The training environment is shown below:

- Hardware
  The device used in the experiment was Dell T480 with i5-8350U 8 cores CPU @1.70GHz 1.90GHz, 16GB RAM.

- Deep learning framework
  The model was trained in Keras [1] framework with the backend of Tensorflow 2.0 [2]. The word2vec model was from gensim [3]. The data was trained with a mini-batch of 1024. I used the Adam optimizer with learning rate $\epsilon = 10^{-3}$ and categorical cross-entropy as the loss function. Accuracy was also calculated as an extra evaluation matrix. The hidden dimension of LSTM was 128 and the models were trained for 10 epochs

- dataset
  There are 497 normal event log sequences of successful tests and 29 abnormal event log sequences of failed tests. The set of normal sequences was split to 90% and 10% as training set and validation set respectively. The set of abnormal sequences will be used to test the anomaly detection performance of the model.

[1] https://keras.io/
[2] https://www.tensorflow.org/
[3] https://radimrehurek.com/gensim/models/word2vec.html

The model was trained on normal sequences in the training dataset to predict the next events after the input sequences. To acquire a model with the relatively best performance, I compared the prediction performance of the attentional LSTM models and the bidirectional attentional LSTM models with window sizes T of 16, 32, 64 and embedding dimensions dw of 16, 32, 64 as shown in Table 5.1. The results of our models were compared with the LSTM model in Deeplog [Lin *et al.*(2016)Lin, Zhang, Lou, Zhang, and Chen] with the same configurations. When the accuracy scores of two models are similar, the one with lower loss is better.

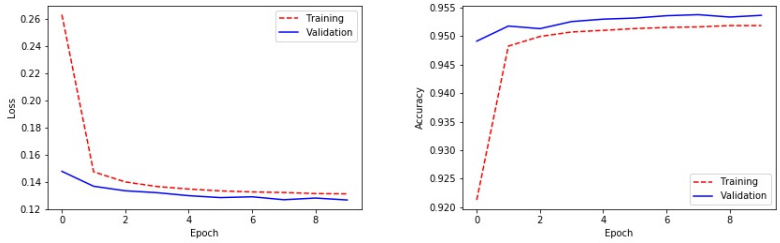
Model	T	dw	Training		Validation	
			Loss	Accuracy	Loss	Accuracy
Deeplog	16	64	0.2026	0.9047	0.2011	0.9052
	32		0.1650	0.9455	0.1634	0.9457
	64		0.1377	0.9503	0.1344	0.9519
		32	**0.1361**	**0.9510**	**0.1310**	**0.9532**
		16	0.1380	0.9507	0.1343	0.9519
Attentional LSTM	16	64	0.1988	0.9333	0.1974	0.9340
	32		0.1573	0.9455	0.1571	0.9479
	64		0.1479	0.9557	0.1471	0.9559
		32	**0.1424**	**0.9496**	**0.1362**	**0.9516**
		16	0.1491	0.9480	0.1441	0.9503
Attentional BiLSTM	16	64	0.1928	0.9343	0.1897	0.9354
	32		0.1598	0.9549	0.1570	0.9457
	64		0.1343	0.9530	0.1305	0.9529
		32	**0.1328**	**0.9554**	**0.1216**	**0.9589**
		16	0.1342	0.9515	0.1289	0.9538

Table 5.1: A comparison of the prediction performance of models using different parameters.

In terms of the results in Table 5.1, the selection of window size has more effect on the training performance than embedding dimension. The larger window size the model uses, the longer memory the model learns so that the result becomes better. But meanwhile, the training time takes longer. For all kinds of models, their best results were given by the models using the window size of 64 and embedding dimension 32 while the best result among all models was given by the attentional bidirectional LSTM model where the validation loss and accuracy after 10 epochs were 0.1216 and 0.9589. The learning curves of this model are shown in Figure 5.1.

5.2. ANOMALY DETECTION AND DIAGNOSIS

Since the model is trained on normal sequences only, given a new sequence, events in the sequence that deviates from the model are likely to be anomalies. The degree to which an event does not conform to the model is determined by the anomaly score. In this section, I compared the performance of different anomaly score thresholds on the validation dataset and testing dataset. Moreover, I evaluated the anomaly detection results using the best threshold.



(a) Loss (b) Accuracy

Figure 5.1: Loss and accuracy curves of the model with best performance

5.2.1. ANOMALY SCORE THRESHOLD

The difference between the prediction and the actual event is measure by the anomaly score. To compare the performance of different anomaly score threshold, I use the model with the best performance to detect anomalies from the normal sequences in the validation dataset and the abnormal sequences in the testing dataset. The threshold is determined by minimizing the number of detected anomalies in the normal sequences. The losses from normal sequences have a mean of 0.7954 and a standard deviation of 3.4666. I experimented the threshold with $(mean + std * n)$ where n = 0, 1, 2, 3. In terms of Table 5.2, it is clear that the larger threshold gives better result and the threshold of 11.1936 gives the best result of 98.77% precision and 0.01% false positive rate. Keeping increasing the threshold might give us even better result but also take a higher risk of losing important anomalies. Therefore, we take the 11.1936 as the anomaly score threshold.

Threshold		Precision	False positive rate
mean	0.7954	0.3337	0.0379
mean + std	4.2615	0.6674	0.0041
mean + std * 2	7.7281	0.9416	0.0004
mean + std * 3	11.1936	0.9877	0.0001

Table 5.2: The performance of different thresholds

5.2.2. ANOMALY DETECTION RESULTS

Using the trained model and the anomaly score threshold of 11.1936, I detected anomalies from the event logs of failed tests in the testing dataset. However, since there are many unknown patterns and correlations among events in such a complicated test, it is impossible to know all the root causes of failures and their related events. Therefore, due to the lack of ground truth, an effective approach of evaluating detected anomalies is proposed. For the detected anomalies of each event log, we manually examine whether they are important to the root cause of the failure or not. If yes, it is a true anomaly; otherwise, it is a fake anomaly. The evaluation was done by the expert in ASML based on his experience and judgment.

For the 29 event logs of failed tests in the testing dataset, we detected and evaluated anomalies of 25 event logs because the other 4 event logs are too short (shorter than the window size) for the model to learn. As shown in Table 5.3, for all anomalies detected by the model, around 73% of them were important and related to the root causes of the failures. I also divided the number of total anomalies by the number of event logs as shown in Table 5.4. This result means that, for each event log in the average length of 60000, using the anomaly score threshold of 11.1936, our model can detect around 10 events that have the highest anomaly scores and 7 out of 10 are verified to be important to the root causes of failures. And the time spent on reviewing these 10 anomalies given by the model is definitely shorter than the time spent on reviewing the original event log which includes 60000 events. These results prove that our model can effectively detect crucial anomalies from the event log of a complicated test with a detection accuracy.

We also reviewed the results when lower thresholds were used. We found no new important anomaly but much more fake anomalies detected in the sequences. That means that lowering the threshold from 11.1936 will not help detect more useful anomalies but only decrease the detection accuracy.

True anomalies	Fake anomalies	Total anomalies	Detection accuracy
187	68	256	0.7305

Table 5.3: Anomaly detection results of 25 event logs of failed tests

True anomalies	Fake anomalies	Total anomalies	Detection accuracy
7.48	2.72	10.24	0.7305

Table 5.4: Averaged anomaly detection results

5.3. VISUALIZATION

In this section, a graphic user interface is designed to help users intuitively localize and understand the anomalies. The interface integrates the score plot, attention map, and raw event logs. The user can have a global view of the anomaly distribution and zoom in to see details. An event logs window on the right top displays the raw log text descriptions of the query event and the prior event with the highest attention score in default. It can be updated when the user clicks other prior events.

5.3.1. ANOMALY SCORE GRAPH - DETECTING AND LOCALIZING THE ANOMALY

An example of using the threshold 11.1936 is shown in Figure 5.3(a)5.3(b). The scores of most events in the normal sequence are zero while there are some clear peaks in the score plot of the abnormal sequence. A high anomaly score indicates an anomaly and a cluster of anomalies indicates a collective error. The user can zoom in to see the detailed plot of the cluster as shown in Figure 5.3(c).

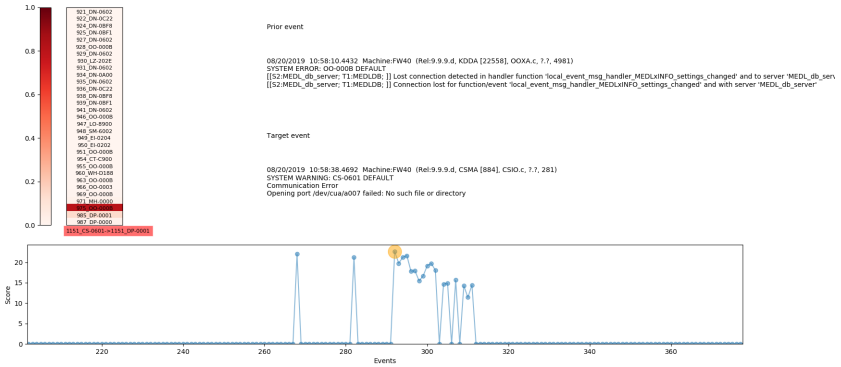


Figure 5.2: Visual interface overview

5.3.2. ATTENTION MAP - UNDERSTANDING THE ANALYSIS PROCESS OF THE MODEL

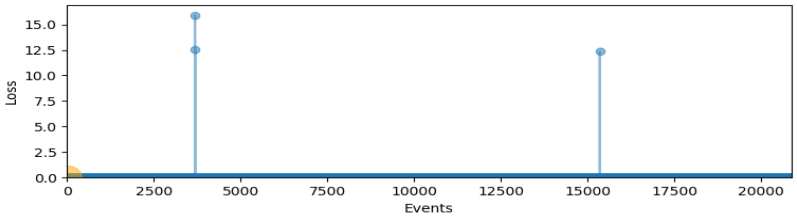
I will explain the usage of the attention map by an example. The event CS-0601 (marked by an orange circle in Figure 5.3(c)) has a relatively high anomaly score compared to other events. By clicking that point on the anomaly score plot, I can have the attention map for its input sequence which was used to predict this event and raw log descriptions as shown in Figure 5.4. According to the attention map, the last three prior events have relatively high attention scores while event OO-000B has the highest. In terms of their raw event logs shown in Table 5.5, the *lost connection* event OO-000B caused the *communication error* reported in CS-0601 while the other two events did not directly relate to the query event. This proves that an event is not always related to its closest event. Also, according to the anomaly score plot 5.3(c), there is a cluster of anomalies following this event. So, this event is likely to be the root cause of the following anomaly cluster.

Event	Log description
OO-000B	Lost connection detected in handler function ...
DP-0001	Wrong parameters.
DP-0000	Internal software error.
CS-0601	Communication Error.

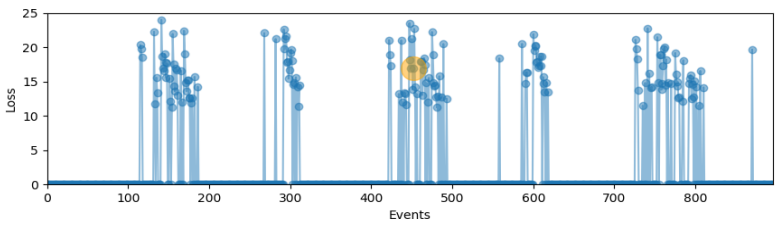
Table 5.5: An anomaly example

5.3.3. SUMMARY

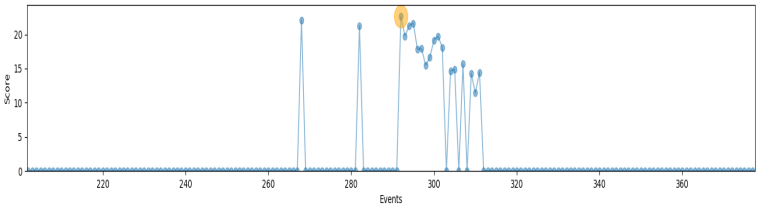
The visual interface provides the user a new interactive way of diagnosing the event log instead of reviewing the raw event log text file. With the help of this visual interface, the user can localize the anomalies and understand the relations between anomalies and normal events in a short time.



(a) Normal sequence

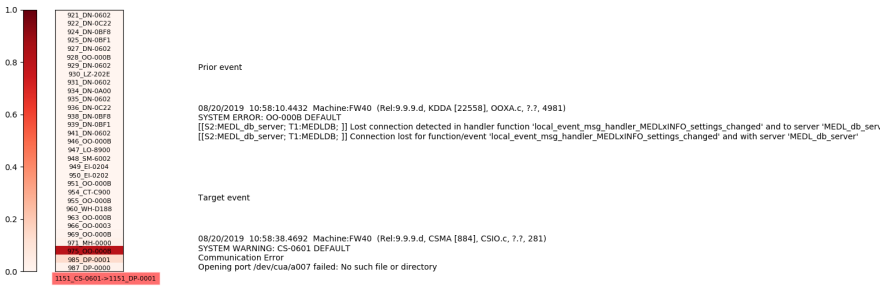


(b) Abnormal sequence



(c) Zoomed in plot

Figure 5.3: Score scatter plots of a normal sequence example (top) and an abnormal sequence example (bottom).



(a) Attention map

(b) Raw event logs

Figure 5.4: Attention map and raw event logs

6

DISCUSSION

In this chapter, I answer the research question proposed in the first chapter. Moreover, I analyze the current limitations of my work and some thoughts.

6.1. ANSWERING RESEARCH QUESTIONS

Now, the research questions proposed in the first chapter can be answered.

RQ1: How to format the unstructured log data and select the feature?

By consulting experts and observing the data, I end up using the event code to represent the event in event logs with the reason that the event code can summarize the most information in the event message. The timestamp is an attribute commonly used to monitor the status of runtime but discarded in this paper because the ASML log data are generated irregularly and there are several events functionalized to alarm the timeout activity. So, each event log file can be parsed to a set of event codes. This way can reduce the size of each file form around 1.5MB to 140kB which effectively speed up the loading of data during training.

RQ2: How to clean the log data?

Due to the lack of domain knowledge of me, I had no idea what events are noise and can be filtered from the data. Therefore at the beginning, I had a hard time because of the low classification accuracy and the large number of fake anomalies. So, I spent a lot of time on diagnosing the reason why they were incorrectly detected as fake anomalies. That was because:

- They do not have causality with their nearby events.
- They are the duplicated events of prior events.
- They are in the execution loops.

I therefore proposed several filters to remove these noisy events from the dataset. These filter are applicable to any ASML event log data. Using these filters can not only clean the data and reduce the false positives but also can reduce the length of the event log. For example, the average length of event logs of successful tests can be reduced from 60000 to 20000 after data cleaning.

RQ3: The dataset has around 600 types of events and each log has around 60000 events in average. These events are known to be more or less causally related to their adjacent events. How to robustly model the causal dependencies among events in such a big and complicated dataset?

An attentional LSTM model was proposed to model the normal sequences in the training dataset. The long sequences were separated into many short sub-sequences using a sliding window. I use the last events of sub-sequences as outputs and the rest of the events as inputs. Given a new sequence, the model can predict the next event under the premise of prior events. The performance of the model can be improved by increasing the window size. With the window size of 64, we can have the model with 0.1216 validation validation loss and 0.9589 validation accuracy which are better than the performance of the baseline model proposed in Deeplog (0.1310 validation loss and 0.9532 validation accuracy). Moreover, the interpretability of the model was also improved a lot by introducing the attention mechanism.

RQ4: Given an event log file of a failed test, how to detect, localize and interpret anomalies?

Given a new sequence, we can use the model to predict future events. If the actual event is different from the prediction given by the model, this event is likely to be an anomaly. We can have a global view of the locations of anomalies by observing the anomaly score plot. To open the black box of the deep LSTM neural network, an attention layer was appended to measure the importance of elements in the input on the prediction of output. Events in the input with relatively high attention scores are considered to be more relative to the query event. In this way, we can understand the analysis process of the model and the cause of the anomaly. A visual interface integrating anomaly score and attention map was built to help the user use the model.

6.2. LIMITATIONS AND THOUGHTS

One of the biggest challenges of this paper came from the complexity and the large scale of the dataset. Due to the lack of knowledge of ASML semi-conductor and software testing, it was very difficult for me to understand the content of test logs and effectively rule out the noisy events from the log. Unlike other sequence modeling tasks such as Natural Language Processing (NLP), understanding the log sequence of a test and reviewing the anomalies detected by the model needs rich software testing experience. Also, it is impossible to re-run the test and compare the status of the test before and after removing the anomalies. All the above difficulties made the process of model optimization and evaluation very challenging

THE FAKE ANOMALIES

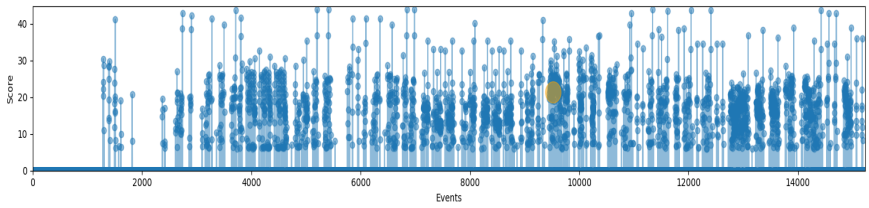
The fake anomaly in the section is defined as the anomaly detected in an abnormal sequence but not relate to the failure of its corresponding failed test. Using my model could help reduce the size of the anomaly candidate pool from thousands to tons that save users a lot of time on diagnosing the event log. However, there are still some fake anomalies and the ratio of important anomalies is sensitive to the selection of the anomaly score threshold. When using a high threshold, only the most dangerous anomalies will be detected but some related events might not be lost. Oppositely, using smaller threshold

can contain more relevant information but also more fake anomalies.

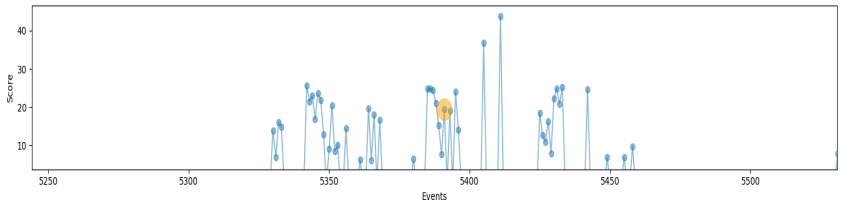
One reason for the above problems may be that there are still some unknown noise events in the data set and the model is distracted by these noises when extracting the real pattern from the sequence. To adequately model such a complex test also requires more training data while removing as much noise as possible and shortening the length of each sequence. Although the data set we used was already the largest test case in ASML, it was only 526 long sequences, which might not be enough to solve the deep learning problem.

THE RESOLUTION OF VISUALIZATION

After applying several data filters, I manage to shorten the average length of sequences from 60000 to 20000. However, it is still a large number of data for visualization. When there are too many points in the graph, the graph is easy to get messy as shown in Figure 6.1(a). Even though there are only a few hundred anomalies detected from a sequence in the length of 15000, the user may feel that there are anomalies full of the screen and difficult to find the part to focus on. Also, it gets difficult to click the exact point due to the clutter. Unfortunately, the capability of displaying a clear graph is limited by the number of points in the graph. The graph becomes readable after a few zoom-in as shown in Figure 6.1(b).



(a) The score plot with clutter



(b) The score plot after a few zoom in

Figure 6.1: An example of clutter in score plot

We have a similar problem on the attention map as well. In the Figure 6.2, an attention map with the window size of 64 was plotted. When the window size increases, the length of the attention map becomes longer. However, because the place for placing the attention map is limited, the annotations of units in the attention map become unreadable. The user can read the detailed text description of each event in the event log

module on the right side of the interface by clicking that unit.

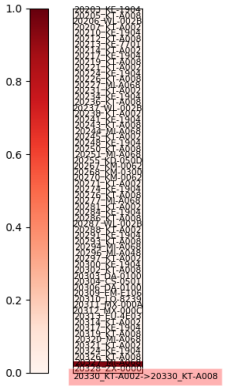


Figure 6.2: An example of clutter in attention map

Therefore, in the future, I would like to explore another appropriate visualization approach that can adapt to a large number of data and build a smoother user interactive interface.

7

CONCLUSION

7.1. CONCLUSION

In this paper, I propose a pipeline to process and clean the ASML unstructured event log data. Several filters are designed by myself and applied to remove the irrelevant events and shorten the length of the event log. More importantly, besides modeling the event log sequence by the long-short term memory recurrent neural network, I introduce the attention mechanism which is very popular in the Natural Language Processing to anomaly detection and analysis. The proposed model has been proved to effectively detect important anomalies from the event log. Compared to manually review and diagnose the root cause from the event log in the averaged length of 60000 which is usually time consuming, the approach proposed in this paper can efficiently diagnose the event log in a few seconds. Also, a visual interface is built to provide the user another interactive way of diagnosing the event log.

7.2. FUTURE WORKS

The anomalies detected from the events logs using our model were evaluated by the expert of ASML. The feedback received from the users can be used to optimize the model. One way could be reviewing what events are easily examined as fake anomalies and giving higher weights to these events when predicting.

A test is an instance of a test task. The model of this paper was trained and tested only on a collection of tests of one task. As different test tasks might have different characteristics, in the future, this approach can be applied and tested on other test tasks. Moreover, as the dataset used in this paper was downloaded from TestArena, more tests of this task might be available in the future. The model can be updated when a new test is executed. Also, when more test tasks are available, we can train a largest word2vec model on all tests in ASML. So the model can include all types of events and capture more accurate dependencies among all events in different contexts.

Recently, there are some state-of-art algorithms in Natural Language Processing such as Transformer [Vaswani *et al.*(2017)Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosu

Instead of using RNN or related recurrent model, these algorithms proposed a brand new way to model the dependencies in the sequence and learn the attention weights of every input. They have shown powerful performance on solving machine translation problems. In the future, a comparison of the performance of Transformer and Recurrent network on modelling event log sequence can be studied.

REFERENCES

[Akidau *et al.*(2015)Akidau, Bradshaw, Chambers, Chernyak, Fernández-Moctezuma, Lax, McVeety, Mills, T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, *et al.*, *The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing,* Proceedings of the VLDB Endowment **8**, 1792 (2015).

[Liang *et al.*(2007)Liang, Zhang, Xiong, and Sahoo] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, *Failure prediction in ibm bluegene/l event logs,* in *Seventh IEEE International Conference on Data Mining (ICDM 2007)* (IEEE, 2007) pp. 583–588.

[Chuah *et al.*(2010)Chuah, Kuo, Hiew, Tjhi, Lee, Hammond, Michalewicz, Hung, and Browne] E. Chuah, S.-h. Kuo, P. Hiew, W.-C. Tjhi, G. Lee, J. Hammond, M. T. Michalewicz, T. Hung, and J. C. Browne, *Diagnosing the root-causes of failures from cluster log files,* in *2010 International Conference on High Performance Computing* (IEEE, 2010) pp. 1–10.

[Lin *et al.*(2016)Lin, Zhang, Lou, Zhang, and Chen] Q. Lin, H. Zhang, J.-G. 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* (ACM, 2016) pp. 102–111.

[Goldberg and Levy(2014)] Y. Goldberg and O. Levy, *word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method,* arXiv preprint arXiv:1402.3722 (2014).

[Mikolov(2013)] T. Mikolov, *Word2vec,* https://code.google.com/archive/p/word2vec/ (2013).

[Mikolov *et al.*(2013)Mikolov, Sutskever, Chen, Corrado, and Dean] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, *Distributed representations of words and phrases and their compositionality,* in *Advances in neural information processing systems* (2013) pp. 3111–3119.

[Rumelhart *et al.*(1988)Rumelhart, Hinton, Williams *et al.*] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, *Learning representations by back-propagating errors,* Cognitive modeling **5**, 1 (1988).

[Hochreiter(1998)] S. Hochreiter, *The vanishing gradient problem during learning recurrent neural nets and problem solutions,* International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **6**, 107 (1998).

[Hochreiter and Schmidhuber(1997)] S. Hochreiter and J. Schmidhuber, *Long short-term memory,* Neural computation **9**, 1735 (1997).

[Olah(2015)] C. Olah, *Understanding lstm networks,* https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (2015).

[Du *et al.*(2017)Du, Li, Zheng, and Srikumar] M. Du, F. Li, G. Zheng, and V. Srikumar, *Deeplog: Anomaly detection and diagnosis from system logs through deep learning,* in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (ACM, 2017) pp. 1285–1298.

[Xu *et al.*(2009)Xu, Huang, Fox, Patterson, and Jordan] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, *Largescale system problem detection by mining console logs,* Proceedings of SOSP'09 (2009).

[Zhou *et al.*(2016)Zhou, Shi, Tian, Qi, Li, Hao, and Xu] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, *Attention-based bidirectional long short-term memory networks for relation classification,* in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (2016) pp. 207–212.

[Graves and Schmidhuber(2005)] A. Graves and J. Schmidhuber, *Framewise phoneme classification with bidirectional lstm and other neural network architectures,* Neural networks **18**, 602 (2005).

[Bontemps *et al.*(2016)Bontemps, McDermott, Le-Khac *et al.*] L. Bontemps, J. McDermott, N.-A. Le-Khac, *et al.*, *Collective anomaly detection based on long short-term memory recurrent neural networks,* in *International Conference on Future Data and Security Engineering* (Springer, 2016) pp. 141–152.

[Vaswani *et al.*(2017)Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need,* in *Advances in neural information processing systems* (2017) pp. 5998–6008.