Log File Anomaly Detection

Tian Yang
NVIDIA Inc.
tiyang@nvidia.com

Vikas Agrawal
NVIDIA Inc.
vagrawal@nvidia.com

Abstract

Analysis of log files pertaining to a failed run can be a tedious task, especially if the file runs into thousands of lines. Using the recent development in text analysis using deep neural networks, we present a method to reduce effort needed to analyze the log file by highlighting the most probably useful text in the failed log file, which can assist in debugging the causes of the failure. In essence we reduce the log file by removing the lines which are found to be of less importance to the debug. We measure the accuracy of our reduction by F1 score, as well as custom scoring method on manually labeled data, where important lines from a 'failed' log are extracted by the experts of the domain.

1 Introduction

Anomaly detection in log file deals with finding text which can provide clues to the reasons and the anatomy of failure of a run. Most commonly, domain and tool specific regular-expression from previously seen error messages is used to dig the relevant text from the log file. However such regular expressions need persistent manual management to ensure the correctness and relevance. Also newer messages of failures are easily missed with this method.

Conversely, a black-list regular-expression file, which contains regular expressions for all the un-harmful 'text' is also sometime maintained. However such list can grow huge fast, in our domain of vlsi-design, to thousands of lines. This makes further maintenance and execution practically prohibitive.

Ahmed [1] used grammar inference to find anomalous text. Here a representative grammar is constructed out of training set, and compared against the test set to figure text which doesn't conform to the learned grammar.

Similar to above, our hypothesis on log file anomaly detection relies on the fact that any text found in a 'failed' log file, which looks very similar to the text found in 'successful' log file can be ignored for debugging of the failed run. Anomaly detection is trying to find 'salient' or 'unique' text previously unseen.

In this work, we propose to use neural network language model learned from 'successful' log files, to predict the anomalies in a 'failed' run, there by attempting to reduce the size of the log file which needs manual review. With no domain knowledge or tokenization, we claim to get upto 85% log size reduction, while keeping more than 77% of useful information. The reduced log file contains most of the text relevant to the failure of the run, with additional text which is considered false positive. The text which is useful for debug, but missing from reduced log file is considered as false negative. We propose to use F1 score as measure of accuracy of the reduction.

2 Model description

Language models for prediction of next word based on contextual statistical data has been in existence for many years with incremental improvements of n-gram models [2]. More recently, "Recurrent Neural Network [RNN]" based statistical model [4,5] have been able to provide better language model and more forgiveness for non-exact (n-1) phrase in an n-gram. Further, invent of character vector based RNN-language model [6,7] reduces the vocab size and allows better learning from text which has high variability tokens [such as date-stamp and runtime metrics].

We experimented with word vector based RNN language [word-RNNLM] model as well as char vector based language RNN model [char-RNNLM] as described in the following section.

2.1 Word Vector based Language model [word-RNNLM]

We used 1 hidden layer recurrent neural network for training.

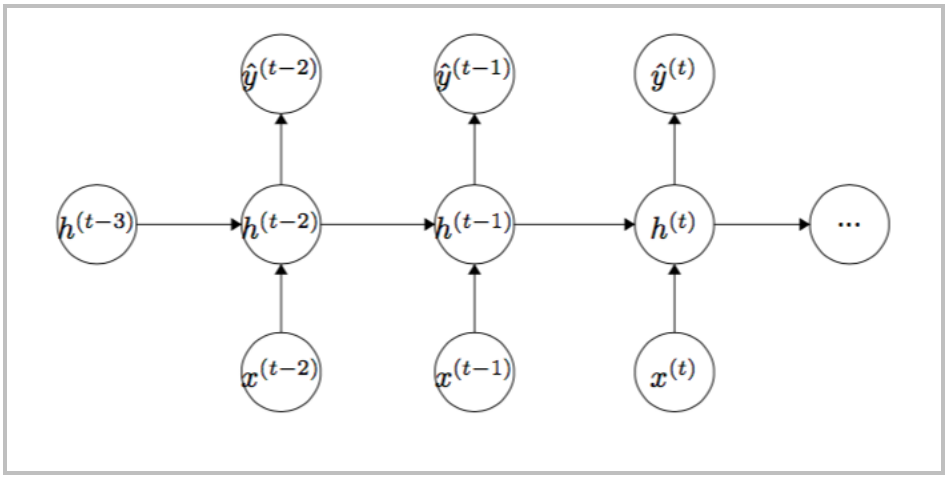

Figure 1: Structure of one hidden layer RNN used in word vector based language model

Figure 1 shows the structure of word vector based language model. Here $x^{(t)}$ is the input word vector (of dimension) at t timestamp, $h^{(t)}$ is the hidden layer (of dimension), $\hat{y}^{(t)}$ is the predicted probability of next word at t timestamp:

$$\boldsymbol{h^{(t)} = \text{sigmoid}\left(h^{(t)}H + x^{(t)}I + b_1\right)} \quad \ldots\ldots (1)$$

$$\boldsymbol{\hat{y}^{(t)} = \text{softmax}\left(h^{(t)}U + b_2\right)} \quad \ldots\ldots (2)$$

Cross entropy loss function was used for training:

$$\boldsymbol{J(\theta) = \text{CE}\left(y^{(t)}, \hat{y}^{(t)}\right) = -\sum_{i=1}^{|V|} y_i^{(t)} \log \hat{y}_i^{(t)}} \quad \ldots\ldots (3)$$

During testing, the objective was defined as to find the paragraph loss of each paragraph. Each paragraph contained just one sentence delimited by newline character in the dataset we used.

Sentence loss hence was defined as the average loss RNNLM model found by addition of each word in the sentence.

$$\boldsymbol{loss = \frac{\sum_{t=w+1}^{L} J_t(\theta)}{L - w}} \quad \ldots\ldots (4)$$

$J_t(\theta)$ is the cross entropy loss at t timestamp, L is the number of tokens in the sentence,

and w is the number of tokens we used for warm up the hidden state during testing.

2.2 Character Vector based Language model [char-RNNLM]

We used recurrent neural network of 2 hidden layers with hidden layer dimension of 128 and sequence length of 50 and 150. We experimented with GRU, RNN and LSTM based models with various parameters.

Sentence loss function in char-RNNLM was defined as the average loss of each character. Same loss function (4) is used with L representing the total number of characters in the sentence and w being the number of characters for warm up during test.

3 Experiments

3.1 Dataset

We collected 90K log files from various stages of the VLSI design process, and sampled 2.5K files with uniform random distribution based on length, process, and design attributes. The log files were of varying length from 500 lines to 20000 lines in length.

3.2 Experiments with word vector RNNLM

Since log file contains several token which are not in English vocabulary, number of unique tokens quickly grow with each file added into training data. without any tokenization we have 116,000 words in vocabulary, with about 2500 log files, totaling to 810,000 lines. Reduction in Vocab size was pretty much essential, and we did aggressive tokenization of log file data, splitting it for all special characters, and merging all numbers into a single token. Post tokenization, Vocab size reduced to somewhat manageable 14K words.

Without tokenization, training took about 48 hours, which reduced to 6 hours with tokenization on a TITAN X GPU with 1100 Mhz clock. Most experiments with word vector RNN were limited to better tokenization due to large vocabulary size.

Dataset is divided into Training, Dev and Test with 50/25/25 split. Language model was trained with final parameters of [learning rate=0.001, dropout=0.5, hidden_size=200, embed_size=50]. Final training perplexity [=1.62], Validation perplexity [=2.14], and test perplexity[=2.18] was achieved on the language model.

3.3 Experiments with Char-RNNLM

3.3.1 Training raw data

We trained the char-RNN on a reduced database containing roughly 150,000 lines, [to limit the training runtime] and used a dev-set of 60 labeled log file and tested the model performance on 40 labeled log files. For the character RNNLM we didn't use any tokenization and used the raw log file data as it is. We did however experiment with tokenization for one model as described in next section.

Training took between 10-12 hours on the TITAN X GPU with 1100 Mhz clock, for most of the experiments.

3.3.2 Addition of some domain knowledge

Analysis on the false positive indicated a few common trends. One of them was the difficulty of learning with variable tokens such a 'block-name', which shows up in lines of text such as

*Value 'DESIGN(UPF_FILE,ICC)' changed by ./**NV_FBIO_hs_wck_AFCR90**.yaml: from 0 to 1*

Typically, these lines contained text of format:

<***block_name***>.<***file_attribute***>.<***file_extention***>

We used following two regular expressions to tokenize such 'words'

s!<\S+>\.\([^]*\)!BLOCK.\1!g

Which results in:

Value 'DESIGN(UPF_FILE,ICC)' changed by ./BLOCK.yaml: from 0 to 1

3.5 Measurement

Goal of anomaly detection is to remove unimportant lines from a failed log file, such that reduced log file contains all the useful information needed for the debug of the failure. For the purpose of dev/test, we manually reduced a set of 100 log files, to minimal size which contained all the useful information about the failure. We split it into 60/40 as dev/test set.

For each of the trained model, we calculated sentence loss for each line of the dev set, and given the sentence loss for each line, we calculated the threshold loss [$\boldsymbol{Loss_{th}}$], such that mean F1 score across all dev set is maximized. We used this threshold to reduce the test-log file size. Any lines with $loss < \boldsymbol{Loss_{th}}$ were suppressed. We then calculated F1 score of the test set, along with precision and recall. We report the mean and std-dev of F1 score, across all test logs, as well as precision, and recall in the section 4.

$$Loss_{th} = \text{argmax}_{loss \in \{loss_s | s \in dev\}} F1(Loss) \ldots\ldots (5)$$

The reduction (%) R is defined as:

$$R = \frac{\sum_{i=1}^{N} 1\{Loss_i < Loss_{th}\}}{N} \times 100\% \ \ldots\ldots (6)$$

Where N is the total number of lines in the "failed" log file and the numerator is the number of lines remaining below the sentence loss threshold.

4 Results and analysis

4.1 Choosing RNN Model

Table 1, below shows the comparison of accuracy and log file reduction as seen by different RNN models we experimented with.

Method	Seq. length	R%	Precision	Recall	F1 score	
					Mean	Std-dev
Word-RNN	50	74.15%	0.123/0.135	0.445/0.228	0.170	0.153
Char-LSTM	50	87.8%	0.525/0.381	0.570/0.317	0.361	0.321
Char-RNN	50	75.3%	0.471/0.32	0.652/0.192	0.507	0.299
Char-GRU	50	73.4%	0.407/0.319	0.637/0.186	0.446	0.298

Table 1: Comparison between RNN models

Word-RNNLM suffered from low precision though compared to all three char-RNN based methods, though comparable reduction of log file. For the sequence length of 50, simple RNN based model worked best in terms of F1 score. Further we focused mainly on char-RNN models with different cells [gru, lstm and rnn], for its versatility.

4.2 Effect of Tokenization on char-RNNLM

Note that for all the models we tried to achieve the best F1 score, and report the reduction and accuracy parameter for the loss-threshold, which achieves the best F1 score.

Table 2 shows the effect of small amount of tokenization as explained in section 3.3.2. Tokenization with domain knowledge helped improve the F1 score, however achieved ~10% lower log file reduction.

Model	Seq. length	R%	Precision (mean/std-dev)	Recall (mean/std-dev)	F1 score	
					Mean	Std-dev
LSTM	50	87.8%	0.525/0.381	0.570/0.317	0.361	0.321
LSTM-token	50	77.2%	0.392/0.362	0.698/0.221	0.403	0.318

Table 2: Tokenization for char-RNN

4.2 Effect of Sequence length on char-RNNLM

Table 3 shows effect of sequence length variation on different cells of char-RNNLM.

Model	Seq. length	R%	Precision (mean/std-dev)	Recall (mean/std-dev)	F1 score	
					Mean	Std-dev
RNN	50	75.3%	0.471/0.32	0.652/0.192	0.507	0.299
RNN	150	15.2%	0.085/0.162	0.979/0.032	0.130	0.034
RNN	250	13%	0.063/0.081	0.95/0.022	0.109	0.133
LSTM	50	87.8%	0.525/0.381	0.570/0.317	0.361	0.321
LSTM	150	90.3%	0.379/0.297	0.555/0.199	0.420	0.282
LSTM	250	74.4	0.415/0.323	0.609/0.243	0.446	0.327
GRU	50	73.4%	0.407/0.319	0.637/0.186	0.446	0.298
GRU	150	78.88%	0.416/0.317	0.756/0.146	0.473	0.297
GRU	250	85.6%	0.597/0.238	0.775/0.160	0.661	0.215

Table 3: Sequence length

Simple RNN starts well with sequence length of 50, with best F1 score (0.507), amongst the three cells, however as the sequence length is increased in hope for better F1 score and log file reduction, GRU outperforms with highest F1 score and a reasonable good reduction in log file size. We see that its able to retain > 77.5% of the important text (as seen in the recall value), while reducing the log file by 85.6%. GRU and LSTM have better performance when sequence length getting larger, gradient vanish problem become severe for regular RNN.

4.3 F1 score maximization:

Figure 1 shows the variation of F1 for the best model [GRU/250], plotted against the loss threshold.

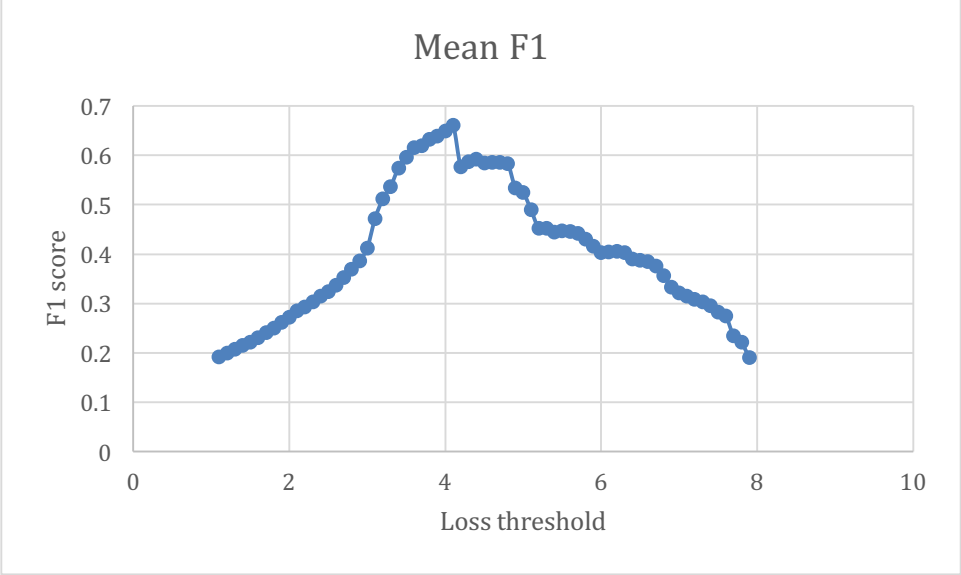

Figure 1

5 Conclusions and future work

The fundamental hypothesis of this work, concerning ability to differentiate the text which is similar to previously during training Vs newer text, through RNN based language modeling worked quite well. We were able to show reduction in log file by upto 85% using raw log files, while keeping more that 77% of useful information intact. For the smaller sequence length, addition of domain knowledge in terms of tokenization helped improve the accuracy, however even better accuracies could be achieved by simply increasing the sequence length, and using GRU cell.

Analysis on false positives reveal a few trends, such as words which are used only in one or two unique phrases tend to get high loss in the during testing. Also that token which contain domain specific information, which is subject to change each run, sometimes results in high loss.

Not all "newer" text is of same importance in debugging the log file. And the next step is to train the language model using the additional text from "failed" log files which is found to be not useful for debug, however it's somewhat new. Addition of such text into training model is expected to further compress the final log file size.

We also believe that using the language model, and suitable training data, we can remove the high variability tokens such containing information such as 'block name' and 'design environment variables', and then they can be replaced by fixed tokens. This should improve the performance of the anomaly detection significantly.

Acknowledgments

We would like to thank Richard Socher and the course staff [cs224d-2016], for useful insight and suggestions during this work.

We would like to acknowledge use of char-RNN modeling code based on Andrej Karpathy's char-rnn, by the Github contributor Sherjil Ozair [3].

References

[1] Ahmed Umar Memon (2008). "Log file categorization and anomaly analysis using grammar inference" . Master thesiss, Queen's University Kingston, Ontario, Canada

[2] Goodman Joshua T. (2001). A bit of progress in language modeling, extended version. Technical report MSR-TR-2001-72.

[3] https://github.com/sherjilozair/char-rnn-tensorflow

[4] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[5] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur, (2010). "Recurrent neural network based language model," in Proc. Interspeech, , pp. 1045-1048.

[6] Cicero D. Santos and Bianca Zadrozny, (2014). “Learning Character-level Representations for Part-of-Speech Tagging”, Proceedings of the 31st International Conference on Machine Learning (ICML-14).

[7] Kim, Yoon, et al. (2015), "Character-aware neural language models." arXiv preprint arXiv:1508.06615.