CS230

DeepBug: A hybrid of CNN and RNN approach for software bugs triage

Yuanliang Lu
Motorola Mobility Inc.
lofus@stanford.edu
lofus@motorola.com

Lucy Gao
IBM Inc.
lgao@ibm.com

Dawny Liu
Motorola Mobility Inc.
dawny@motorola.com

Abstract

We propose a deep neural network approach named as DeepBug for software bugs triage. DeepBug takes the bug report title, description and logs as input features, and triage that to which software component the bug belongs to. Deepbug is designed as a hybrid architecture with a CNN and a RNN running in parallel. The CNN path keeps learning from the bug log file in a way mimicking manual log analysis by layers, and the RNN path works on learning the text sequences from bug title and description. The stream of the two paths are aggregated as the final output as a classifier. Validated by both public and industry dataset, DeepBug demonstrated an exciting performance on the benchmark of a BOW model such as [15] [17] [21], and the RNN model as by Senthil et. al. [24].

1. Introduction

Software bugs are usually managed with a system like JIRA. Each bug report comes with attributes including title, description and log files. We have been spending a huge effort on the bugs triage and assign it to the right team for further analysis and get it resolved. In this project we explored some state-of-the-art deep learning technologies and built a RESTful service called DeegBug, which works to triage the software bugs automatically and precisely. When a new ticket comes into JIRA system, DeepBug service predicts that to which software components might belongs to. The predictions serve as a guideline to move it forward, and reach the right component as early. A high level view for the DeepBug system design is shown as Figure 1.

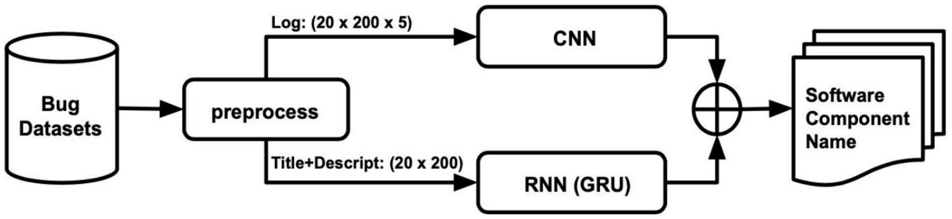

Figure 1: DeepBug system architecture

This design is based an insight that our software architecture is essentially a stack with multiple layers and the bug report log is actually a snapshot to the stack run-time. For a given layer in the stack, the log tends to have a segment with some distinct symptom in the trace. When expertise analyze the logs he will review the log prints by layers, look for the exceptions and identify to which layer, which component the bug is happening. So the intuition is that we can take a CNN to learn the bug's log stack in the way mimicking manual log analysis by people, the process could also be similar to image recognition, such as to extract and learn the features from the logs for different layers of the software stack, and abstract from a high level (later cnn layers) to classify for the bug to which software component (labeled) it has happened. In the other hand, the title and description for a bug are descriptive text sequences, and we can take a RNN model to learn the features from the text context. At the end we aggregate the results from CNN and RNN and softmax it as the final output. From CNN and RNN path the data stream comes with same dense after flatten and right before softmax, so we can merge them up. The output and the cross entropy loss can be expressed as (1) and (2), though the implement are mostly with Keras frameworks.

$$\tilde{y} = softmax(C + R) \tag{1}$$

$$L(y, \tilde{y}) = -\Sigma_i y^i \log \tilde{y}^i \tag{2}$$

This design of DeepBug is adaptive. For these bugs without a log trace, the path of CNN model is actually ignored, so it falls back to the path of the RNN model, which is similar to the approach by Senthil et al. [24] We firstly trained a CNN model (simple as AlexNet) for classification by bug's log trace, and separately build a RNN path. We compared the RNN model with GRU v.s LSTM units, and selected GRU approach which has a better performance of 86.25% accuracy for top 20 components. Another interest work, we replaced the RNN path with a CNN model, it works to classify the bugs with only title and description, but the overall performance is not good as RNN.

2. Related Work

Bug triage is a classification problem, and most of recent related work [15] [17] [19] [21] [25] took the bug summary/title and description as input features, and learn to identify an appropriate developer as the owner. We observed that various approaches available in literature, based the models such as tf, normalized tf, tf-idf, and n-grams. Xuan et. al., 2015 [17] used tf, and feature selection with naive Bayes, and got an accuracy of 60.40%. S Kim et. al [25] used a tossing graphs model which achieved a accuracy around 70%. These existing projects are essentially deploying a bag-of-words (BOW) feature model, and learning to map the bug title and description to one of the developers (class labels). BOW does not absorb well the syntactical and sequential information available in the unstructured text. Bhattacharya et. al. [21] used additional attributes such as product, component, and the last developer activity to target the developers, and the accuracy is around 74%. More recently we have seen some exploring for deep learning models for this task, Senthil et. al.,[24] proposed an algorithm based on a deep bidirectional recurrent neural network model which learns the syntactic and semantic features from the text description, but the bug logs is not taken into account. In this project, instead classify for developer we challenge to triage for software component, and we propose a model with a hybrid CNN and RNN running in parallel, target to learn from both descriptive input and the log file, which is a structured text information.

3. Datasets and Features

We initially trained DeepBug with public dataset for bug records from project **Google Chromium** and **Mozilla Firefox**, which comes with a total of 240K samples. Each sample is composed with a bucket of bug title, description and owner, where owner is the email ID of the developer who had resolved the bug. This dataset turns out not working well, and we switched to train and validate DeepBug with the industry dataset and our internal data for bug reports from JIRA system with a total of 200K samples, where we explicitly set the feature 'owner' to be the software component name against which the bug was resolved. We abstract the title, description, log and component as the key inputs for a given ticket report. The data features and label are listed as following.

- **Title**-- This is a summary for the bug, usually it is less than 30 words.
- **Description**-- A description of what are the scenarios and behaviors from user experience and or testing observation. This could be more than 300 words.
- **Logs**-- This can be an attachment such as log files, which is a snapshot to the software stack, a printed from each layer of the Android software stack.
- **Component** -- This is the actual ***data label***. For a closed ticket, this is the software component name where the bug was finally fixed. For a new ticket, this means the component name that DeepBug works to predict for.

The datasets are organized up as .json files. For each bug, the title, description and component features are wrapped up within a JSON object. The log trace are preprocessed and get wrapped with a separate JSON object. The .json files are cleansed, such as removing the tabs, the hex code and URLs. We removed these because they do not contain any valid descriptive next. The text is then embedded and vectorized with Word2Vector and feed into the models. The whole data pipeline is illustrated as Figure 2.

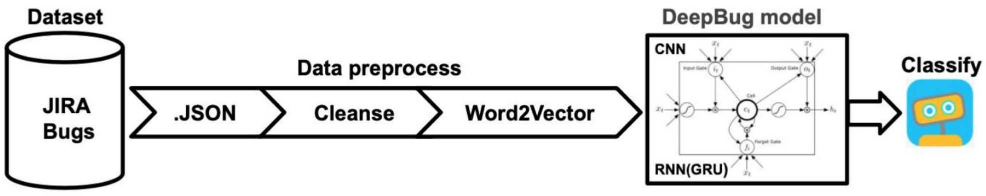

Figure 2: The data pipeline

In this pipeline, the vocabulary is pre-compiled with a whole dataset for all the bugs on known components, and for training purpose we filter out the bugs for top 20 components. The title and description are vectorized as (*20 × 200*) dimensional vector. The log file processing is different. We first split the log tracce into fragments according to the layers of the stack stack. In this project the software stack is based Android platform and the logs typically contains 5 layers, known as Android application layer, framework layer, RIL/OEM layer, Android kernel layer, and baseband layer. Log fragments are vectorized and we stack them up as *20×200×5* dimensional matrices.

4. Methods

With Keras framework we built and trained for three models: 1) CNN for logs learning, 2) RNN models with LSTM, and 3) RNN with GRU which learn from title and description. The goal is to classify the new bugs, in another word, it is to predict the probability to which software component the bug might belongs to. We start the CNN model prototype with an AlexNet design, it constructs with three convolution layers, get concatenated with a flatten layer, and lastly a softmax for output. In this model the inputs are *20 × 200 × 5* matrices and we applied a padding initially and then conv2D(). The CNN model generally works for classifying the bugs for the given 20 components. We also tried the same CNN model for the dataset of bug title and description, where we reshaped the input from *20 × 200* to be *20 × 200* × 1, and interestingly it also works, though the performance, overall accuracy 73.54%, is not so good as compared to

RNN model for that path, learn by title and description. Intuition is that we are actually treating the text vector as an one channel image, and it can not learn well for the text context information, so we switch to try out RNN models. Result shows the RNN(GRU) model achieved a better accuracy performance over the other 2 models. Initially we trained and validated the CNN and RNN models separately, with public datasets from Chrome and FireFox bug repos, where the 'owner' feature in the dataset are actually the developers' email IDs. And it turns out the algorithm does not converge well on the training dataset, and performance is poor, that the prediction accuracy oscillates at 45%. Our analysis suggest that there is no correspondences in between the bug and the assignee who had ever worked on that bug and fixed it. Instead, there is a relation between the software module (we usually call it as component in each layer of the software stack) and the bugs fixed against it. We switched the training dataset from industry and our internal JIRA database, where we managed to set the 'owner' feature to be the component name, against which the ticket was ultimately resolved. Our dataset contains 200K samples, and the learning converges well after 200 epochs of training. We have chosen adam as the optimizer for both CNN and RNN paths, and used a batch size of 1024. A plot of training loss per epoch is shown as Figure 3.

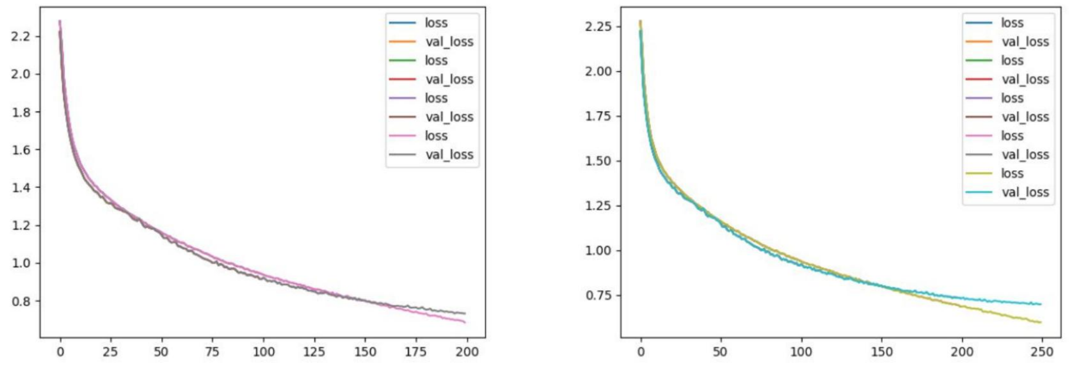

Figure 3: The loss for 200 and 250 epochs

5. Results

Validated by the industry datasets and our JIRA dataset, DeepBug has achieved an accuracy of avg 86.25% for the top 20 software components. Figure 4 shows a benchmark with the accuracy as by SME manual triage, which is generated by a big query into our JIRA dataset.

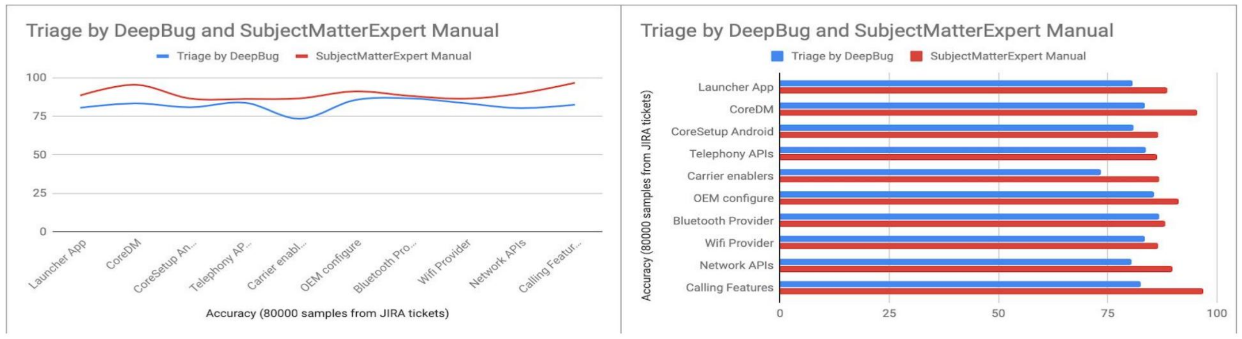

Figure 4: Triage accuracy: DeepBug vs SME manual

For the RNN path we compared the performance for RNN(GRU) and RNN(LSTM) models, and suggest RNN(GRU) reached a better performance for top10 components cross validation and the average accuracy on test set. Figure 5 shows a plot of the cross validation result for this.

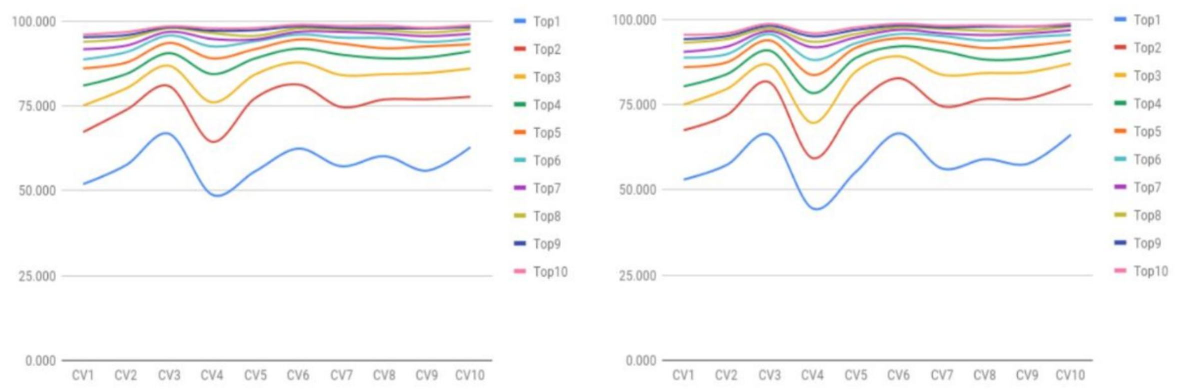

Figure 5: CV result for RNN(GRU) and RNN(LSTM)

6. Discussion

6.1 Classify for software component instead of developer ID. Initially we trained and validated the CNN and RNN models with open source Chrome and FireFox bug dataset, it turns out learning does not converge well and performance is not good, such as the prediction accuracy oscillates at 45%. With analysis into the dataset we concluded that in nature there is no distinct connection in between a developer and the bug fixed by him, because a developer might be working on some component for a while, and change his working domain then. Instead, there could be a connection in between the software component name and the bugs fixed against that component, or by that component team. Base on this intuition we switched to our JIRA dataset and set the owner to be component, and it is proved that this mapping works well, DeepBug model reached an accuracy of 86.25% and even with a single CNN model it can get a 75.34% accuracy for triaging the bugs for top 20 components.

6.2 Switch from CNN to RNN model. We also tried a CNN model for classifying the bugs in the path with bug title and description. The learning curve converges and we got an average 75.34% accuracy on test set. Though there is a dramatic oscillation in the loss as observed in the first 50 epochs, with a learning rate 0.0001, as shown in Figure 6. Intuition is that we are actually treating the text vector as an one channel image, the text vectors could have a variance in between difference components, so we reshuffled the dataset and it works to smooth out the oscillation, but the accuracy remains at same level (< 76.00%). To have a model which learns well for the text information especially for the context, we switched to RNN models for this path.

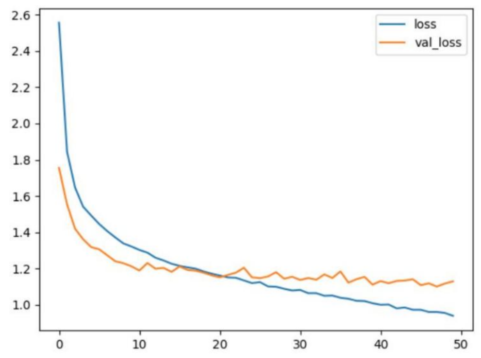

Figure 6: CNN model for title-description path

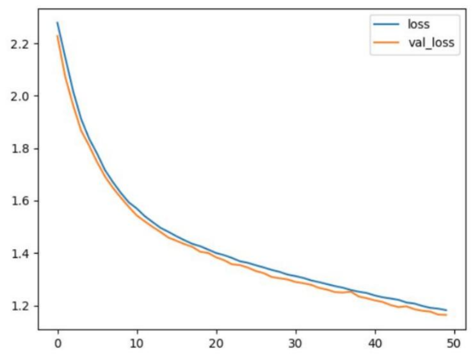

Figure 7: validation loss and training loss

6.3 Resolve the running gap in between validation loss and training loss. The variance happens to be high in our early tuning, and we deployed dropout(keepprop=0.6) for each layer and the variance issue got resolved, however we noticed there was a running gap inbetween validation_loss and the training_loss from the beginning. A plot of loss illustrates the loss gap issue as figure 7, we can see the val_loss keeps running below training loss, in another word the validation loss looks better than training loss. This is not expected and our analysis suggest it is due to excessive using of dropout. It dropped out quite neurals while calculating the training loss yet a full network weights calculate the validation loss. We then decreased dropout keep-prop from 0.6 to 0.4, and have Dropout only for the last layer before Softmax dense, it resolved the running gap issue.

7. Conclusion and future work

DeepBug model has demonstrated to be promising for our software bugs triage, with an accuracy at 86.25% for average on the top 20 components. We plan to scale Deepbug for all 300 components, it could save a huge effort which is currently done by subject matter experts (SMEs) triaging manually. DeepBug can be improved to support for an even big data space, as in this project the DeepBug model, especially in the CNN path it is actually trained for a software system where the log file is built like a stack, where it has several distinct layers and corresponding logs. It might not generalize well for a different software architecture such as the logs does not comes with traces as per layers.

8. Contribution and Acknowledgments.

In this project Dawny has worked on the analysis and design of a service which works to pull the dataset from JIRA and convert it into .json files. Lucy from IBM contributed the analysis for industry bug report structure, query with big data and generate the benchmark for manual accuracy. Lucy helped to validate the DeepBug model with different dataset. Specially we would like to thank our mentor Suvadip Paul for the helpful discuss and valuable advice such as for text vectorizing, developing deep neural network with Keras frameworks, and tuning for hyperparameters for deep learning models such as CNN and RNN.
Huge thanks to CS230 instructors, Andrew Ng and Kian Katanforoosh, for such a great course.
The source code and test reports can be found at **https://github.com/lofus/deepbug**

9. References

[1] Zhang X, Zhao J, Lecun Y, et al. Character-level convolutional networks for text classification [C]. Proceedings of the 29th Annual Conference on Neural Information Processing Systems, 2015.
[2] Kim Y. Convolutional Neural Networks for Sentence Classification [C]. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'14), 2014: 1746-1751.
[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.
[4] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In INTERSPEECH, 2014.
[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.
[6] Johnson R, Zhang T. Supervised and semi-supervised text categorization using LSTM for region embeddings [C]. 33rd International Conference on Machine Learning (ICML'16), 2016: 526-534.
[7] Zhang S, Zheng D, Hu X, et al. Bidirectional Long Short-Term Memory Networks for Relation Classification [C]. Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation (PACLIC'15), 2015: 73-78

[8] Pamela Bhattacharya and Iulian Neamtiu. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In International Conference on Software Maintenance, 2010.
[9] Wang, C. Manning. Baselines and Bigrams:Simple, Good Sentiment and Topic Classification. In Proceedings of ACL 2012.
[10] Yang, C. Cardie. Context-aware Learning for Sentence-level Sentiment Analysis with Posterior Regularization. In Proceedings of ACL 2014.
[11] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In International Conference on Software Engineering, pages 404–415, 2016.
[12] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025, 2015.
[13] John Anvik, Gail C Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. ACM Transactions on Software Engineering, 2011.
[14] Zhang T, Yang G, Lee B, et al. A Novel Developer Ranking Algorithm for Automatic Bug Triage Using Topic Model and Developer Relations [C]. Proceedings of the 21st Asia-Pacific Software Engineering Conference (APSEC'14), 2014, 1: 223-230.
[15] Cubranic D, Murphy G C. Automatic bug triage using text categorization [C]. Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering, 2004.
[16] Ahsan S N, Ferzund J, Wotawa F, et al. Automatic Software Bug Triage System (BTS) Based on Latent Semantic Indexing and Support Vector Machine [C]. Proceedings of the Fourth International Conference on Software Engineering Advances (ICSEA'09), 2009: 216-221.
[17] Jifeng Xuan, He Jiang, Yan Hu, Zhilei Ren,Weiqin Zou, Zhongxuan Luo, and Xindong Wu. Towards effective bug triage with software data reduction techniques. IEEE Transactions on Knowledge and Data Engineering, 27(1):264–280, 2015.
[18] Yang G, Zhang T, Lee B, et al. Towards Semi-automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-feature of Bug Reports [C]. Proceedings of the IEEE 38th Annual Computer Software and Applications Conference (COMPSAC'14), 2014: 97-106.
[19] Philip J Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In International Conference on Software Engineering, volume 1, pages 495–504, 2010.
[20] Ali Sajedi Badashian, Abram Hindle, and Eleni Stroulia. Crowdsourced bug triaging. In International Conference on Software Maintenance and Evolution, pages 506–510. IEEE, 2015.
[21] Bhattacharya P, Neamtiu I, Shelton C R, et al. Automated,highly-accurate, bug assignment using machine learning and tossing graphs [J]. Journal of Systems and Software, 2012, 85(10): 2275-2292.
[22] Anvik J, Murphy G C. Reducing the effort of bug report triage: Recommenders for development-oriented decisions [J]. ACM Transactions on Software Engineering and Methodology, 2011, 20(3): 10:1-10:35.
[23] Park J, Lee M, Kim J, et al. COSTRIAGE: a cost-aware triage algorithm for bug reporting [C]. Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI'11), 2011: 139-144.
[24] Senthil Mani, Anush Sankaran, Rahul Aralikatte, Exploring the Effectiveness of Deep Learning for Bug Triaging, 2016
[25] G. Jeong, S. Kim, T. Zimmermann, Improving Bug Triage with Bug Tossing Graphs, in: FSE, 2009.