SIMILAR TASK PREDICTION WITH SIAMESE RECURRENT NEURAL NETWORK

[1]MUSTAFA ERŞAHIN, [2]KAZIM KEMAL EMRE AKI

Commencis Technology , Commencis Technology
E-mail: [1]mustafa.ersahin@commencis.com, [2]emre.aki@commencis.com

Abstract - There are several companies in different industries have their own task lists. In general, these tasks are randomly assigned to people. Sometimes a task is assigned to an employee who do not have enough experiment on task detail. Therefore, it causes having unsatisfied customers. We propose a similar task prediction method to assign tasks to the right employee according to information about previous tasks. Task summary, description and employee's comments are the major information that we feed our proposed Siamese recurrent neural network. Our experimental results which based on software development and maintenance show us, we have quicker bug-fix time than we had before.

Index Terms - Neural Networks, Task Prediction, Text Similarity

I. INTRODUCTION

Digital transformation is one of the main topic in industry nowadays. Traditional manufacturing and industrial practices are combined with technological world that around us. There transformation effects both companies and end users. People want to reach any kind of service easily also they want to be satisfied after the service. Every service provider on each sector have to adapt their own practices or processes to give more standardized, modern service. Therefore, it is getting higher number of service providers transform their processes to digital channels. There are lots of kind data which is collected from different channels with transformed systems. Volume of the collected data is getting higher, so data is getting more important day by day. Therefore, the data should be analyzed. Analysis of the data helps to people on many ways such as; finding improvement areas, employee performance evaluation, future predictions etc. Each of these areas are important for service providers to have better service quality and customer satisfaction. Quality assurance is one of the major process for software companies. Every commercial application that is developed for customer has provide to some quality metrics. In general, bugs/defects are the major issues that make people think about the software as low quality. Everyone wants to use bug-free application or they want to prioritized support for fixing bugs that they reported. Software development is a combination of processes that should be done in order. This order defined as Software Development Life Cycle(SDLC). This lifecycle includes phases such as; Feasibility Study, Requirement Analysis, Software Design, Software Development, Test and Maintenance.

We focused especially software development, test and maintenance phases to improve our process' efficiency. Bug-fix process directly related with all these processes. Bugs can be reported by different stakeholders such as end users, software testers or developers. After bug production steps are clearly defined, developer reproduce it on local or test environment. Then developer finds the exact code block which causes the bug. Finally developer fixes the bug.

Our motivation is to find the most similar bugs that fixed before to get any valuable information about both cause and solution. After we got the information, it is possible to assign the bug to the right developer to fix it. We think that if we have right assignments, we can fix bugs faster than as usual.

There are two different approaches to find text similarity. One of them is lexical similarity. In Lexical similarity [1] provides the similarity on the basis of character and statement matching. Lexical similarity is a measure of the degree to which word set of two given string are similar. A Lexical similarity of 1(means 100%) would mean a total overlap between words, Whereas Lexical similarity of 0 means there are no common word in given string. The other one is semantic similarity. Semantic similarity [2] determines the similarity between text and document on the basis of their meaning rather than character by character matching. In this paper, we will introduce our proposed method to find most similar bug report in detail.

II. RELATED WORK

One of the greater challenges in Machine Learning, and overall in Artificial Intelligence methodologies is that it is inherently hard to persist some sort of reasoning about the past when making a prediction/classification regarding the current state of the problem. When trying to predict the next occurring word in a text, the model somehow has to know about the previous ones, hence the context.

Likewise, if we were to label objects in a video clip, one frame at a time, it would be much more efficient if we could somehow persist information from the earlier frames to better predict what is going to happen next, rather than starting from scratch at each frame. All of which are similar shortcomings that traditional Neural Network architectures suffered from. The original implementations of Neural Networks, as well as Convolutional Networks were unaware of the context of the problem domain. Not only that, but they also all suffered from some similar constraints—namely, operating over fixed-sized data with a fixed number of computational steps. In conventional Neural Networks, the data that is fed to the network is typically a fixed-size input vector, representing the data points, or samples, in the input set—which is also the case for Convolutional Networks, only this time the input is typically a 3-dimensional vector representing pixel intensities for RGB channels in the input image, separately. These networks carry out their operations for a number of hidden layers, which is determined as a fixed hyperparameter when the model is constructed, and their output is a classification vector that is fixed in size (determined by the number of disparate classes) denoting the probabilities that the input belongs to a particular class in the problem domain.

Recurrent Neural Networks (RNNs), introduced as an improvement over the earlier Neural Network architectures, tackle these issues by wrapping itself in a loop, keeping track of and updating its internal state in each step, influenced by its previous states as well as the current input vector being fed to the network. This way, the model is able to accept a sequence of inputs, as well as being capable of yielding a sequence of outputs. These networks improve input-output mapping to a sequence level (as opposed to fixed-sized input and output), for use in such problems like recognition, prediction and production. Figure 1 shows the basic RNN architecture.

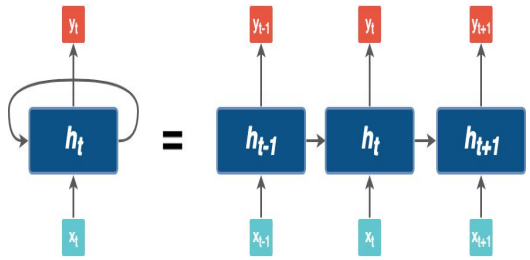


Figure 1. RNN architecture. The blocks in turquoise and orange represent current input and output of the network, denoted by x_t and y_t, respectively. The blocks in between represent the current internal state of the network, denoted by h_t.

As promising as they may seem at persisting and improving upon the past information, when trained using gradient-descent approaches RNN structures have been proven to perform poorly as the dependency gap between the information relevant to the context and the instant it is required grows [1]. In the same paper by Bengio et al, a trade-off was discovered between efficient learning by gradient-descent and losing information regarding the past events over long-term dependencies.

A special kind of RNNs, Long Short-Term Memory Networks (LSTMs) were proposed by Hochreiter and Schmidhuber to overcome the shortcomings of RNNs on long-term dependencies [4]. In their paper, Hochreiter and Schmidhuber introduced new units called gates into the recurrent cell structure to regulate the internal state of the network. These units help the network decide how much of the past and the current state should be passed on to the following iteration via the internal state vector. They do so by putting both the current input vector, and the output vector from the previous iteration through a sigmoid layer before updating the internal state vector of the network. Also, another sigmoid layer is used to determine which components of the past information is to be replaced with the current input, accompanied by a tanh layer to create candidate components from the current input to be replaced with. Figure 2 depicts a single LSTM cell, at time t, in action.

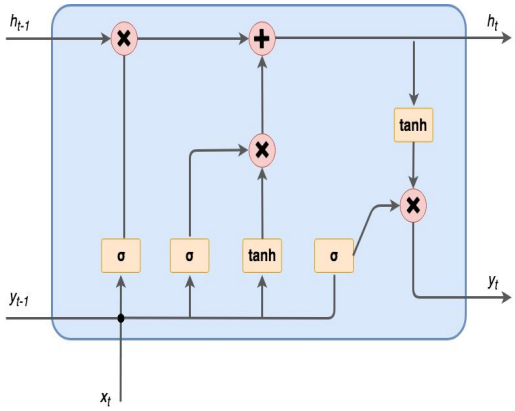


Figure 2. LSTM cell structure, where h represents the internal cell state vector, y represents output vector of the cell, and x represents the input vector to the cell. The red circles denote pointwise operations—multiplication and addition, whereas blocks in yellow represent neural network layers—sigmoid and tanh.

III. ASSESSMENT OF BUGS

Using a Siamese MaLSMT architecture, our aim was to capture semantic similarity between the bug summary and commit message information all retrieved from Jira records regarding various software projects. We then use this mapping between bug summaries and commit messages to assess the probability that a given bug is resolved in a given commit.

Bug summaries are collected from the Description fields of Jira records created as Bugs, whereas commit messages are gathered from various commits done in that particular project. We then pair each bug

summary with commit messages and label them [0, 1], denoting whether that particular bug was resolved in that commit, to form our training data set.

Using a popular Deep Learning API named Keras, we have implemented a Siamese Bidirectional LSTM. We then have fed our model with word embedding vectors, having a dimensionality of 50, generated from bug summaries and commit messages in our training set. We have determined a sequence length of 10 would work best for our case since we had bug summaries of variable lengths for various projects. We have used binary cross-entropy validation during the training process of our model.

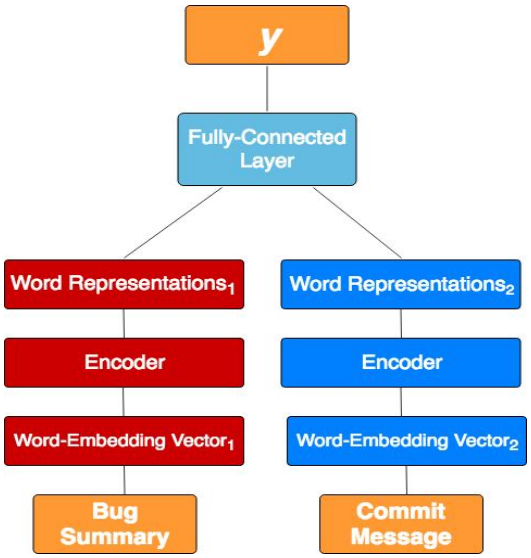


Figure 3. The Siamese LSTM architecture used in our study.

Upon generating word-embedding vectors from text descriptions, we feed them to LSTM encoders to obtain word representations. Finally, these representation pairs are put through a dense layer to achieve a probability, denoting the likeliness that the given bug was resolved in the given commit. Our model can be seen in Figure 3.

IV. EXPERIMENTAL RESULTS

For our experiments, we have collected never-before-seen data from Jira records of various software projects to test our classification accuracy of our model. We have trained and tested our model on four particular project which make up for our bug localization database—AspectJ, Eclipse, FORTE, and SWT.We have trained a single model for each project, and using the Quora data set [5] as a benchmark, we have evaluated our accuracy for each model. The chart below in Table 1 shows model accuracies for each project in correlation with the model trained using Quora data set.

Table 1. The correlation between each indiviual model trained with its respective project data set and the model trained with Quora Semantic Question Matching data set [5].

	AspectJ	Eclipse	FORTE	SWT
CORREL	0.083	0.240	0.080	0.315

CONCLUSION

Along with the digital transformation in many industries, importance of the data has risen drastically. Every company purposes better quality and customer satisfaction. Therefore, most of them change their traditional practices to new or hybrid practices to have more efficient, more productive. In this paper, we propose an approach with Siamese recurrent neural network to find similarity between two sentences. Our purpose is finding the most similar bugs, detailed information about solution according to the commit message. After we find the information, we assign the right developer to fix the bug. We have already experimented the approach on one of our commercial project. The next step is going to apply this approach to the other projects.

REFERENCES

[1] Rensch Calvin R. “Calculating lexical similarity”, In Eugune H. Casad(ed.), Windows on bilingualism, pp:13-15

[2] Sahami, Mehran Sahami, Timothy D. Heilman, “A web-based Kernel Function for Measuring the Similarity of Short Text Snippets”, Proceedings of the 15th International Conference on World Wide Web, pp: 377-386,2006.

[3] Y. Bengio, P. Simard, P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” IEEE Transactions on Neural Networks, vol. 5(2), pp. 157-166, March 1994.

[4] S. Hochreiter, J. Schmidhuber, “Long Short-Term Memory,” Neural Computation, vol. 9(8), pp. 1735-1780, Nov. 1997.

[5] L. Jiang, S. Chang, N. Dandekar. “Semantic Question Matching with Deep Learning,” February 13, 2017, https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning

★★★