A DEEP LEARNING APPROACH TO SOFTWARE CLONE DETECTION

DOAA SHAWKY

Engineering Mathematics Dept., Faculty of Engineering, Cairo University, Giza 12613
E-mail: doaashawky@staff.cu.edu.eg

Abstract - A code clone is a code segment that is similar to another one based on a certain similarity measure. Clones appear as a result of the common programming practice of "copy and paste". However, cloned codes can cause serious problems in software maintenance and upgrading efforts. In addition, they are –sometimes-a sign of poor design. This paper presents an approach for predicting clones in open-source software systems based on the two commonly adopted approaches in deep learning; namely transfer learning and long short-term memory (LSTM) networks. A pre-trained LSTM deep network for text classification is used to extract the relevant features of similar code segments in open source systems. Then a support vector machine classifier is used to classify the clones using the extracted features. Experimental results show that the two employed techniques can predict similar code clones of the used systems with good accuracy, which indicates that the used pre-trained network can detect structural similarity that exists among code clones.

Index Terms - Code clones detection, deep learning, long short-term memory networks, and transfer learning,

I. INTRODUCTION

Code clones are pieces of source code that are similar to those code fragments in several locations [1]. Several studies have shown the harmfulness of clones on software evolution and maintenance. Hence, they should be detected and/or removed [2]. Based on their degree and type of similarity, clones can be classified into three types, which include the following. Type 1 clone is an exact copy without modifications (except for whitespace and comments). Type 2 clone is a syntactically identical copy; only variable, type, or function identifiers have been changed. Finally, type 3 clone is a copy with further modifications; statements have been changed, added, or removed. Sometimes, Type 1 clones are called exact copy clones. Meanwhile, Type 2 and 3 are referred to as near miss clones. Many clone detection approaches have been proposed in the literature. Based on the type of the analysis applied to the source code, the techniques are roughly classified into four main categories: textual, lexical, syntactic, and semantic [3]. In [3], the authors provide a comprehensive comparison of all available clone detection techniques. Some of the syntactic approaches are metric-based i.e. a set of metrics is calculated using the abstract syntax tree (AST) or the control flow graph (CFG) of the code. For example, MOSS (Measure of Software Similarity) [4] which is a free, online service that detects plagiarisms in computer programs, does this by preprocessing the source code files, calculating a numeric fingerprint for each file. Software metrics have proven to provide useful insights about the development process [8]. Hence, there are many works that employed software metrics for modeling and maintenance tasks. For instance, many researchers have used software metrics for fault prediction (e.g., [9]), estimation of software information content (e.g.,[10]), re-engineering or development assessment activities [11-15]. Metrics have also been used in clone detection. For instance, in [5, 6] several metrics are used to identify functions with similar metrics values as code clones. Metric-based approaches have also been applied to finding duplicate web pages and clones in web documents [7].

Transfer learning is the process of utilizing a trained model on some source domain to transfer its knowledge to a new related domain [16]. For example, when using a model that was trained to predict users' preferences on movies in order to predict the suitable advertisements for them, then it is expected that the advertisements recommendations will not be as accurate as movies recommendations. However, the prediction accuracy will reflect partial agreement with users' preferences in this case. The main reason for this is that the two sub-domains are linked in an abstract way. In the proposed approach, a deep learning model for clone detection based on this idea is proposed. Since the main challenge of utilizing deep learning models is the unavailability of a large sample size which is needed to train deep learning models, transfer learning would provide a good solution to overcome this. Using transfer learning, pre-trained deep networks on different-but related- problems can be used, without the need to provide the deep network with a large number of training data. Hence, the power of deep models can be utilized. In this paper, a metric-based code clone detection approach is proposed. The presented approach employs a pre-trained deep network for text classifications based on a set of used software metrics for representing clones. The network architecture is modified to reflect the clones' domain, then it is trained on detecting the

three main types of code clones. Obtained results show that the used network can capture part of the structural similarity that exists among near-miss code clones.
The rest of the paper is organized as follows. Section II introduces a brief background about LSTM networks. In Section III, the experimental study is presented in detail. Section IV presents the conclusions and draws some outlines for the future work.

II. LONG SHORT-TERM MEMORY

A Long Short-Term Memory (LSTM) memory cell is shown in Fig. 1. [17]. These cells enable the network to store information, and hence give the ability to exploit long range dependency.

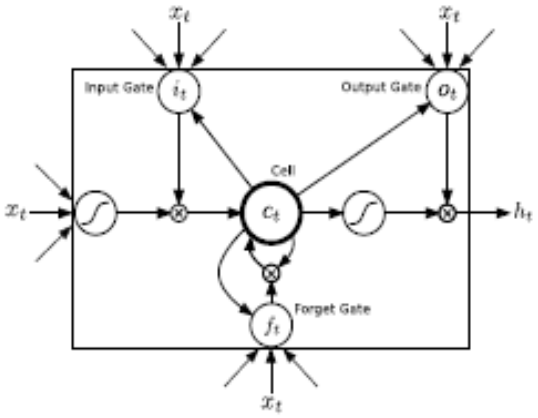


Fig. 1. A memory cell of long short-term memory

III. AN EXPERIMENTAL STUDY

A. Subject Systems and Data Preparation

The used case studies are the two open source systems FileZilla [18] and VLC [19], which are written in C++ and C, respectively. FileZilla is an open source, cross-platform, and FTP client. VLC, on the other hand is a cross-platform open-source multimedia framework, player and server. Table I presents some size metrics about the case studies used.

Table I: Case Studies Metrics

System	Language	KSLOC	# versions
FileZilla	C++	22-105	50
VLC	C	140-145	50

The source files of the systems to be analyzed are parsed using a tool for static analysis [20]. The used level of granularity is at the function level. Thus, a set of six function metrics is used for representing clones. Table II shows the name of each metric as given in the used tool's manual.

The first column indicates the symbols that will be used throughout the paper when the used metrics are referred to.

Table II: Used Metrics

Metric's Symbol	Metric's Name	Metric's Meaning
m_1	CountInput	The number of inputs (parameters and global variables) a function uses.
m_2	CountOutput	The number of outputs.
m_3	CountStmt	Number of declarative and executable statements.
m_4	AvgLineCode	Average number of lines containing source code for all nested functions.
m_5	Cyclomatic	Edges - Nodes + Connected Components.
m_6	MaxNesting	Maximum nesting level of control constructs in the function.

To obtain a set of labelled data for the training stage of the network, the clones in the used systems are detected using the clone detection feature of the Microsoft Visual Studio IDE. The tool generates a list of exact, strong, medium and weak clones. However, only the set of exact and strong matches are used. The set of functions that belongs to clones' groups in each version are used to generate sequences of cloned functions, where only one function from each version is added to the sequence. In total, 230 and 342 sequences of exact matches were generated for FileZilla and VLC, respectively. Meanwhile, 416 and 480 sequences of strong matches were generated for FileZilla and VLC, respectively.

B. Experimental Analysis

The pre-trained network used in the classification of code clones is the Japanese Vowels network, which was trained on classifying a given utterance based on 12 features to one of nine speakers [21]. The dataset on which the network was built contains 640 multivariate time series data points in total, where only a set of 270 data points were used for training and the other set of 370 data points were used for testing. The main reason for using this network is that the domain that is handled by this network is text classification, which is similar to the way we conceptualize our problem of clone detection. As described in [21], the network

architecture is composed of five layers; a sequence input layer with 12 dimensions, an LSTM layer with 100 hidden neurons, a fully-connected layer, a softmax layer, and an output layer which classifies the output into one of 9 classes. Fig. 2 shows the training of the network after modifying the input features to 6 and the output classes to 2. After adapting the weights of the network to that of the clone detection domain, the weights are extracted from the LSTM layer and used as the new features. These sets of features are then used to train an SVM classifier, where the datasets are divided into 80% for training and the rest is used for testing. Table III presents the confusion matrix of FileZilla and VLC. As shown in the table, the average testing accuracies for two systems are 82% and 84%, respectively.

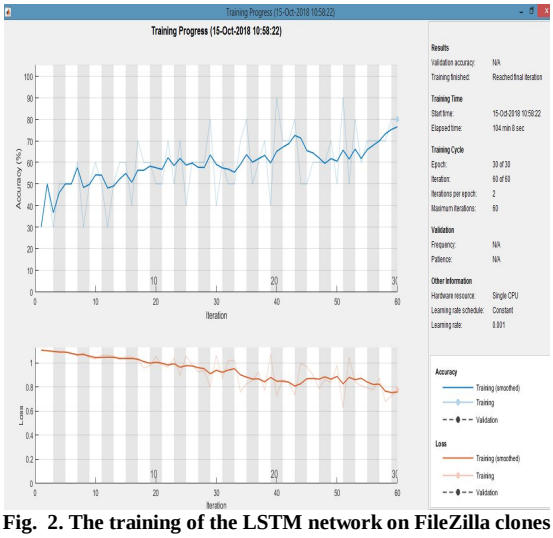


Fig. 2. The training of the LSTM network on FileZilla clones

Table III: Confusion Matrices

		Exact Clones	Strong Clones
FileZilla	Exact Clones	38	8
	Strong Clones	14	69
VLC	Exact Clones	57	11
	Strong Clones	15	81

CONCLUSIONS

Clone detection is an important maintenance and quality verification task, since the addition of clones makes the system more susceptible to errors and inconsistencies, especially during upgrading processes. In this paper, an approach that utilizes transfer learning and LSTM networks for detecting clones is proposed. The clones in consecutive sequences of two software systems are detected and used to adapt the weights of an LSTM layer of a pre-trained network that is used in text classification. The extracted weights are used as the new features. These features are then used to train an SVM classifier with linear kernel, which yielded good classification accuracies on the used systems. Although the sample size is relatively small, obtained good results indicate the suitability of the proposed approach. In addition, the approach sheds the light on some promising deep learning approaches and their applications in software engineering similar to their promising applications in other areas. Future work includes the variation of the architecture of the used network and studying the effect of that this variation might have on the obtained results. In addition to the hyper parameters' optimization of the used classifier.

REFERENCES

[1] N. Hoan Anh, N. Tung Thanh, N. H. Pham, J. Al-Kofahi, and T. N. Nguyen, "Clone Management for Evolving Software," Ieee Transactions on Software Engineering, vol. 38, no. 5, Sep-Oct 2012.

[2] N. Bettenburg, W. Shang, W. M. Ibrahim, B. Adams, Y. Zou, and A. E. Hassan, "An empirical study on inconsistent changes to code clones at the release level," Science of Computer Programming, vol. 77, no. 6, pp. 760-776, 6/1/ 2012.

[3] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," Information and Software Technology, vol. 55, no. 7, pp. 1165-1199, 2013.

[4] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," pp. 76-85: ACM.

[5] D. M. Shawky and A. F. Ali, "An approach for assessing similarity metrics used in metric-based clone detection techniques," in Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on, 2010, vol. 1, pp. 580-584.

[6] D. M. Shawky and A. F. Ali, "Modeling clones evolution in open source systems through chaos theory," in Software Technology and Engineering (ICSTE), 2010 2nd International Conference on, 2010, vol. 1, pp. V1-159-V1-164.

[7] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora, "Identifying cloned navigational patterns in web applications," Journal of Web Engineering, vol. 5, no. 2, p. 150, 2006.

[8] S. M. Cem Kaner, Walter P. Bond, "Software Engineering Metrics: What Do They Measure and How Do We Know?," in In METRICS 2004. IEEE CS, 2013.

[9] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study," Software Process: Improvement and Practice, vol. 14, no. 1, pp. 39-62, 2009.

[10] S. K. Abd-El-Hafiz, "Entropies as Measures of Software Information," presented at the Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01), 2001.

[11] B. Kitchenham, "What's up with software metrics? – A preliminary mapping study," Journal of Systems and Software, vol. 83, no. 1, pp. 37-51, 1// 2010.

[12] D. M. Shawky, "Towards Locating Features Using Digital Signal Processing Techniques," Journal of Engineering and Applied Science, vol. 50, no. 1, pp. 1-20, 2008.

[13] D. M. Shawky and A. F. Ali, "A practical measure for the agility of software development processes," in Computer Technology and Development (ICCTD), 2010 2nd International Conference on, 2010, pp. 230-234.

[14] D. Shawky and S. Abd-El-Hafiz, "The Impact of Agile Approaches on Software Quality Attributes-An Empirical Study," presented at the the 9th International Conference on Software Paradigm Trends, Austria, 2014.

[15] D. M. Shawky and S. K. Abd-El-Hafiz, "Characterizing software development method using metrics," Journal of Software: Evolution and Process, 2015.

[16] S. J. Pan and Q. Yang, "A survey on transfer learning," IEEE Transactions on knowledge and data engineering, vol. 22, no. 10, pp. 1345-1359, 2010.
[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," pp. 3104-3112.
[18] FileZilla, available at https://filezilla-project.org/versions.php.
[19] VLC, available at: https://www.videolan.org/vlc/download-sources.html. VLC.
[20] Understand, a tool for source code analysis and metrics, available at: www.scit.com.
[21] M. Kudo, J. Toyama, and M. Shimbo, "Multidimensional curve classification using passing-through regions," Pattern Recognition Letters, vol. 20, no. 11-13, pp. 1103-1111, 1999.

★★★