International Journal of Current Engineering and Technology
©2021 INPRESSCO®, All Rights Reserved

E-ISSN 2277 – 4106, P-ISSN 2347 – 5161
Available at http://inpressco.com/category/ijcet

Research Article

AST based Semantic Code Clone Detection using Deep Learning

Sudip Mahajan, Dr. Geetanjali V. Kale and Sudarshan Bhide

Computer Department Pune Institute of Computer Technology Pune, India

Received 10 Nov 2020, Accepted 10 Dec 2020, Available online 01 Feb 2021, **Special Issue-8 (Feb 2021)**

Abstract

Software Cloning as an inconsistency in source code has attracted lot of attention across Software Engineering research community. Code Cloning may have adverse effects on software development process and hence a developer should be aware about its facets. Present work aims to highlight challenges in processing Abstract Syntax Tree and applying deep learning approaches on AST representation of source code towards establishing semantic proximity of code fragments. The SeSaMe dataset is used in this work for code clone detection. Results are promising and encourage using larger datasets.

Keywords: *Software clone; code clone detection; deep learning*

Introduction

Computers are an intrinsic part of modern revolution. They have their own language of communication in the form of high or low level programming languages. Source code written using various tools has touched the lives of millions today in varying aspects. Typically programs are written in the form of source code using high level languages which are converted to low level languages using tools like compilers. The Indian IT sector is poised to become USD $ 225 billion industry by 2020. It continues to grow and provide livelihoods to many software developers. Generally, software undergoes various stages before they are launched for intended customers. It is essential to analyze and perform various tests on source code of these soft wares before it is deployed to the market.

Source code(a)	**Type 1 clone(b)**	**Type 2 clone(c)**	**Type 3 clone(d)**	**Type 4 clone(e)**

int main()
{
int x = 1;
int y = x +
5;
return y;
}
	int main()
	{
	int x = 1;
	int y = x +
	5;
	return y; //
	output
	}
		int func2()
		{
		int p = 1;
		int q = p +
		5;
		return q;
		}
			int main()
			{
			int s = 1;
			int t = s +
			5;
			t/++s;
			return t;
			}
				int func4()
				{
				int n= 5;
				return ++n;
				}

Fig. 1. Types of Clones [2]

Code Smell is an indicator or hint that something is not right with the code. Clone detection is a problem of static code analysis used to reduce code smells. Code cloning is the most important code smell described in the literature and it has received due attention in recent times [1]. Research community is yet to find a proper and universal definition of the term 'clone'. Similarly, definitions for types of clones are also hazy in literature with little distinction amongst them. Dominently, 4 major types of clones are identified as shown in Fig. 1. As can be seen from Fig. 1, types 3 and 4 are the most ambiguous types, whereas types 1 and 2 are easy to identify.There are a variety of reason why cloning happens in software development including code reuse, inadvertent mistakes, inconsistent source code editing etc. Code Cloning can lead to software maintenance and management difficulties as observed by considerable studies [3]. Cloning can cause harm to software development process in long run due to vulnerable nature of cloned code. The original code, if containing bugs, can lead to spread of bug infested code due cloning. Hence it is important to be aware of software cloning scenarios and have robust tools for code clone detection.Source code can be seen in the form of many representations such as Abstract Syntax Tree (AST), Program Dependence Graph(PDG) etc. as shown in Table 1. Every representation has its pros and cons. No representation has been shown to be perfect for this task. In present work, the representation of AST is used owing to its widespread use and prolific availability of AST generator tools. AST is an important intermediate representation when source code transforms itself from high level language to machine understandable binary code. It is seen as a condensed form of parse tree generated after syntactic analysis phase of code compilation.

Related Literature

A. CNN based approaches

Convolution Neural Networks are an important class of algorithms which have proved most useful on images.

CNNs are extended versions of simple neural networks with additional capabilities. The ImageNet challenge propelled CNNs to limelight for successful deep feature extraction. CNNs have special layers such as max-pooling, convolution, fully-connected layers to extract higher level features from input. CNNs were traditionally defined on image inputs and encounter difficulties to work upon ASTs. CNNs show capability to extract Deep features but using them on Tree Structures like AST is difficult and needs more exploration. Source code is a non-conventional input for which CNN was NOT made. To modify CNNs right from the definition of convolution is a promising way to use CNNs over source code. Accordingly [4] propose a new architecture called as Tree Based Convolution Neural Network with a custom defined Tree based convolution layer. The method calls are often important in a code snippet to understand meaning of the code. An AST has generic nodes to indicate method invocation which causes loss of information. Hence, [5] again use Tree Based Convolution over api-enhanced Abstract Syntax Tree for code clone detection problem. The AST used by them contains additional annotations, like invoked APIs, to aid in semantic code clone detection.

Table I basic types of code representations

Category	Code form	Comparison	Clone types
Text based	Text	String matching	Type 1
Token based	Token	Token matching	Type 1, Type 2
Metric based	Text	Metric vector	Type 1, Type 2, Type 3
Abstract Syntax Tree	AST	Subtree matching	Type 1, Type 2, Type 3
Program Dependence Graph	PDG	Isomorphic sub graphs detection	Type 1, Type 2, Type 3, Type 4

B. LSTM based approaches

LSTMs are an improvement over recurrent neural networks with additional gates to prevent vanishing / exploding gradient problems. They have proven successful in capturing long term dependencies in Natural Language Processing and hence may prove useful for source code as well. [6] try to extract lexical and syntactic features from source code in order to establish software functional similarity between codes. The approach is to use LSTM on AST to generate a representation which is hashed to generate a hash value of original source code. Thus hash value can be used for comparison with another code snippet for establishing similarity. Such use of LSTMs on ASTs is full of drawbacks as

1. AST is processed as a sequence of tokens by LSTMs and the use of AST is only as a guide to indicate order of consideration of code tokens. The structure of code is not really captured by above technique.
2. The number of child nodes varies for different nodes and hence AST sometimes needs to be converted into full binary tree.
3. Capturing long term context has always been a challenge for LSTMs. It becomes more important to establish dependencies in the context of code.

C. RNN based approaches

Code can be seen as a sequence of tokens just like text in NLP. Hence Recurrent Neural Networks can be applied on code to capture its meaning. Recursive Neural Networks are a slight variation on Recurrent Neural Networks which visualize code as a sequence of tokens and try to use long range dependencies between tokens. RNNs are composed of cells or stages which are recursive in nature. Every cell has a well defined function that is repeated for all. The sequence token is taken as input, and some output is generated based upon cell logic. At the same time, hidden inputs are passed onto next stage. They have proved important in Natural Language Processing and hence are applied to source code which is seen as formal language. [7] proposed Recursive NN based Siamese networks on ASTs to generate code embedding and compute cosine similarity between generated embedding to determine whether code snippet pair is a clone of each other. [7] combine Lexical and Syntactic features learnt using Recurrent NN and Recursive NN respectively. The challenge with their approach is requirement to convert AST to full binary tree and then to Olive trees because of varying number of child nodes in an AST. Traditional Recurrent Neural Networks suffer from challenges like vanishing gradient and inability to capture long range dependencies between input. They have limited memory and do NOT retain context for a long time.

Proposed Methodology

The broad steps shown in Fig. 2 can be elaborated as follows:

Step 1. Initially, a survey needs to be done of the available and usable datasets. Suitable dataset needs to be identified and studied for the use of code clone detection task. The dataset should ideally contain enough variability and capture realistic scenarios of code cloning during development process.

Step 2. For present work, AST is chosen as a representation of code, hence AST needs to be generated with suitable tools. Different generator tools generate ASTs that vary in the level of detail.

Step 3. The AST generated in step number 2 is referred to as raw AST which cannot be used directly. It needs to be processed and converted into computationally feasible and understandable representation. Here, ASTs are converted into adjacency matrices which are in turn flattened into vector representations.

Step 4. It is now possible to train a deep learning model for code clone detection task. The training phase takes as input code snippets in the form of processed ASTs with associated labels. Here, twin arm based architecture Siamese Network based model is used as shown in Fig 4. The input to the architecture is a labelled code snippet which makes Siamese Networks ideal for code clone detection problem.

Step 5. The trained network predicts whether test snippets are clone or not. Results should be analyzed for further improvement on pre-processing and clone detection logic.

Fig. 3 shows the high level view of proposed architecture used for this task. It is a twin architecture which takes as input a

method clone pair converted into vectors. An explanation of various layers used in the architecture is as follows:

1. *Input Layer:* The role of this layer is to take the input with correct dimensionality and pass it over to Embedding layer.
2. *Embedding Layer:* This layer maps and converts input to vectors of required dimensions.
3. *LSTM Layer:* This stage has only one LSTM cell which takes as input the embedding layer generated vectors. The typical LSTM layer tries to convert input vector to vector of embeddings which captures semantics of input code snippets. In this embedding space, codes which are similar to each other should be close, whereas code fragments of different meaning should move far away from each other.
4.

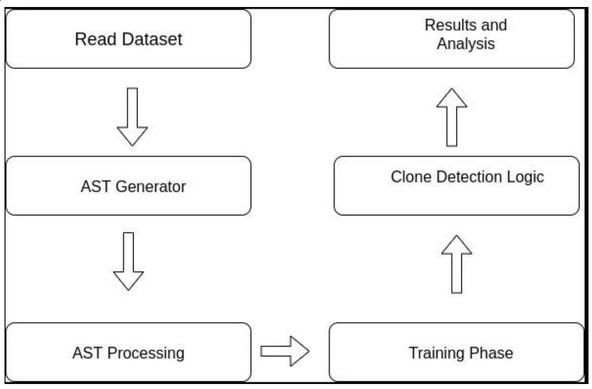

Fig. 2. Methodology for Code Clone Detection

5. Cosine Similarity: This layer computes Cosine Similaritybetween the embedding vectors as generated by previous LSTM layers.
6. *Dense Layer*: This is the final layer consisting of only one neuron which captures threshold of the cosine similarity distance given by previous layer. It decides whether the input code snippet is a clone or not depending upon learned threshold.

Dataset and Experimentation

For present work, the SeSaMe dataset [8] is used. SeSaMe stands for semantically similar Java methods. The authors have used manual tagging as well as algorithmic efforts to establish similarity amongst Java method pairs. They looked at many active project repositories in open domain to ensure the dataset contains actual development code as against synthetic code. The authors assign a similarity score to method pairs which indicates how semantically close the method pairs are. The dataset contains Java method pairs from active repositories which are semantically similar according to following aspects:

1. *Goals* of the method
2. *Operations* contained inside the method
3. *Effects* of the method on rest of the code

It is reasonable to use above points to establish semantic similarity of Java methods. Above points make the dataset suitable to be used for semantic code clone detection problem. However, there are no other approaches making use of SeSaMe dataset for clone detection problem as yet.

In present work, a subset of this dataset is extracted for training and test (cross-validation) purposes. Currently, a subset of 400 method pairs is extracted for this purpose. These method pairs are converted to ASTs which are further converted to adjacency matrices. These adjacency matrices are flattened into vectors for passing them to proposed deep architecture. The ASTs are converted to adjacency matrices using a dictionary of all possible internal types encountered in Java language. Adjacency matrices are converted to vectors by flattening the matrix. About 350 method pairs are used for training and the rest are used for validation purposes. In all, the proposed architecture leads to 16, 418 tunable paramaters to play with. Training happens using Gradient Descent algorithm.

Many Deep learning frameworks are availabel today which are open source and under active development. Implementation of proposed Siamese Network is done using *Keras*, a python based framework owing to widespread use. *Python* is chosen as the language of implementation as it supports many built-in functionalities and packages. The implementation took place in stages of pre-processing, training and testing phase.

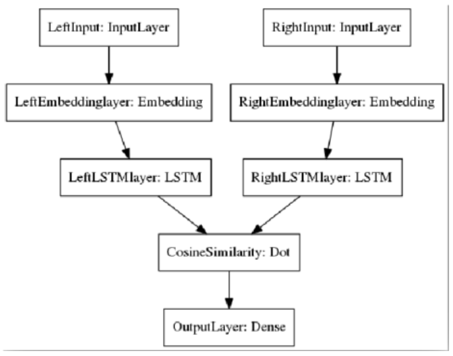

Fig. 3. Siamese Architecture

Results and Discussion

The SeSaMe dataset consists of only 900 pairs of semantically similar java methods from various active repositories, out of which 400 pairs are successfully extracted for this task. Due to limited dataset, the accuracy does not go beyond a certain point. The 'binary cross-entropy' loss does reduce to a minimum of 0.2889 as observed in 5 epochs. The dataset named as BigCloneBench [10] is currently the best known benchmark dataset for the task of code clone detection. Future work consists of training the proposed architecture on such large scale dataset to achieve realistic results and accuracy.

Conclusion

The problem of code clone detection is very important from industrial point of view. Cloning has diverse effects on software maintenance as it can lead to bug propogation. This work used Abstract Syntax Tree representation of code and deep learning networks for identifying semantically similar code snippets. Specifically the LSTM based Siamese Neural Networks show promise as can be seen by reduction in error value on SeSaMe dataset. Code clone detection is still an open problem with lots of practical challenges and ambiguity with regard to important aspects of source code. This work highlights challenges of processing code in the form of Abstract Syntax Tree and building code detection logic using deep learning. For deep learning networks, it is essential to have large scale datasets to learn robust features capturing semantics of data. Future work includes scaling of training dataset to achieve more accuracy.

Acknowledgement

This work is sponsored by Embold Software Private Limited.

References

[1]. A. Gupta, B. Suri, and S. Mishra, -A Systematic Literature, Int. Conference on Computational Science and Its Applications, pp. 665-682, July 2017.

[2]. E. Kodhai and S. Kanmani, -Method-level code clone detection for java through hybrid approach, Int. Arab. J. Inf. Technol., pp. 914-922., Jan 2014.

[3]. M. Morshed, M. Rahman, S. Ahmed, -A literature review of code clone analysis to improve software maintenance process, arXive preprint, May 2012.

[4]. L. Mou, G. Li, L. Zhang, T. Wang, Z. Jin, -Convolutional neural networks over tree structures for programming language processing, Thirtieth AAAI Conf. On Artificial Intelligence, Feb 2016.

[5]. L. Chen, W. Ye, S. Zhang, -Capturing source code semantics via tree-based convolution over API-enhanced AST, Proceedings of 16th ACM International Conference on Computing Frontiers, pp. 174-182, April 2019.

[6]. H. Wei, M. Li, -Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code,In IJCAI, pp. 30343040, Aug 2017.

[7]. L. Buch, A. Andrzejak, -Learning-based recursive aggregation of abstract syntax trees for code clone detection, IEEE 26 th

[8]. Int. Conf. On Software Analysis, Evolution and

[9]. Reengineering. SANER, pp. 95-104, Feb 2019.

[10]. M. White, M. Tufano, C. Vendome, D. Poshyvanyk, -Deep learning code fragments for code clone detection, IEEE/ACM Int. Conf. On Automated Software Engineering . ASE., pp 8798, Sep 2016.

[11]. M. Kamp, P. Kruetzer and M. Philippsen, -SeSaMe: a data set of semantically similar Java methods, IEEE/ACM 16th Int. Conf. On Mining Software Repositories, pp. 529-533 , May 2019.

[12]. J. Svajlenko, J. Islam, I. Keivanloo, C. Roy, M. Mia, -Towards a big data curated benchmark in inter-project code clones, IEEE Int. Conf. On Software Maintenance and Evolution, pp. 476-480, Sep 2014.