TEXT VS TREES VS GRAPHS: DEEP LEARNING TECHINIQUES FOR PROGRAM UNDERSTANDING

MSc Project Preliminary Report

Author: Olubusayo Akeredolu

Email : olubusayo.akeredolu@kcl.ac.uk

Student ID : 20107125

Supervisor : Dr. Maria Polukarov

April 1, 2022

CONTENTS

1. INTRODUCTION

1.1. PROJECT OVERVIEW

This project falls under the field of Programming Language Understanding. This is a field of Artificial Intelligence and Machine Learning that deals with making use of Machine Learning and Deep Learning techniques to train computers, or intelligent agents to understand programs, which will be referred to as source code for the purpose of this project.

Programming Language Understanding is a highly relevant and interesting field because it examines a very specific way in which computers are yet to be as intelligent as humans: they cannot tell the difference between natural language and source code. Consider a situation where a Natural Language Processing (NLP) model [8] is trained using source code. The NLP model is unable to differentiate between an algorithm and a piece of text from a story, and I will demonstrate this using the Recurrent Neural Network (RNN) I am building as part of this project.

There are many differences between natural language and source code; a lot of which computers cannot yet decipher. One of these is that natural language generally requires less context in comparison to source code. For example, a natural language sentence like 'The girl is in the class' makes sense on its own and can be understood without further context. This is not the case with source code. Each line in a program should not be considered on its own due to the nature of programs. For example, the java statement 'a += 10' requires the programmer to consider where the variable 'a' was declared, what was its initial value, under what conditions its value is being changed, etc. The statement does not make sense when considered as a standalone statement and needs to be taken into consideration with the rest of the program.

Another difference is that there is a lot of structural and conceptual information contained in programs that is not contained in natural language. This includes control flow, conditional statements, function and method declarations, and more complex concepts like inheritance, abstract classes, the programming paradigm the program is making use of, etc.

1.2. PROJECT AIMS & OBJECTIVES

1.2.1. MAIN AIMS

The main aim of this project is to compare three machine learning models to assess which is the most suitable for carrying out classification tasks on source code. The first of these three models is a text-based Recurrent Neural Network (RNN) model [10] that considers source code in the same way as standard NLP models. The second is a Tree-Based Convolutional Neural Network (TBCNN) [2] based on the Abstract Syntax Tree (AST) [11] of a program. An AST is the tree representation of a program that shows the structure and connections present in the code. The third model is a Graph Neural Network [4] which will make use of graphs and graph theory to process, understand, and classify source code.

1.2.2. SPECIFIC PROJECT OBJECTIVES

There are 6 main objectives I intend to achieve with this project.

Objective 1: To implement a method of converting source code into a directed graph containing nodes and edges. The nodes will be the various elements in the program and the edges, which will contain information about what is being passed between two nodes, will connect a pair of nodes. This implementation will be based on the methods described in [1].

Objective 2: To utilise the graph generated based on Objective 1 to develop a Graph-Based Neural Network (GBNN) [4] that is capable of carrying out classification tasks on source code.

Objective 3: To implement a method for converting source code into a tree structure. This will be based on the methods described in [2] alongside the use of the Any Python Tree Data library [3].

Objective 4: This objective is based on Objective 3. This is to develop a Tree-Based Convolutional Neural Network (TBCNN) [2] that is capable of utilising the tree generated from Objective 3 to carry out classification tasks on source code.

Objective 5: To develop a simple text based RNN [10] that performs classification tasks on source code using NLP methods.

Objective 6: The final objective is to compare the performances and accuracies of each of the three models to determine which model is most suitable for carrying out classification tasks on source code.

1.3. PROJECT DOMAIN

This project falls under several domains; Artificial Neural Networks (ANNs) [12] which are the foundation for Programming Language Understanding, and Graphs and Graph Theory [9]. My project will involve the application of graphs and Graph Theory in processing, understanding, and classifying programs.

Another important domain is the Tree data structure [13]. Objectives 3 and 4 are completely based on the use of the Tree data structure to process and understand source code. Other domains include Machine Learning, Deep Learning and Natural Language Processing (which is the foundation for Objective 5).

2. BACKGROUND & MOTIVATIONS

2.1. BACKGROUND

This project is inspired by my interest in Programming Language Understanding and my interest in the applications of graphs to programming languages. Every program can be represented as a flowchart or a flow graph, which are in turn forms of graphs. I want to explore the ways graphs can be utilised when processing source code. I am interested in how the high levels of information graphs contain can be used when carrying out learning tasks on source code.

This project is also based on the applications of the Tree data structure to programs. Every program has an AST representation but there is little to no background literature describing the ways in which ASTs can be utilised by neural networks to decipher what a program is doing.

2.2. MOTIVATIONS

The main motivation for this project is the lack of sufficient exploration into the applications of ASTs and graphs for programming language understanding. I am using this project to examine the different ways by which programming language understanding can be made possible.

I am also using this as an opportunity to investigate whether, in the future, computers can be taught – using NNs – to generate meaningful and usable code in the same ways that NLP models can generate meaningful and usable text.

Crucially, this project will examine the way the TBCNN [2] and the Graph Neural Network Model [4] function at the lowest level. I will investigate how they can be utilised for training computers to differentiate between text and source code and how they can be used to understand the underlying structure of a program.

2.3. RELATED WORK

The field of Programming Language Understanding is still relatively new. There are little amounts of pre-existing literature relevant to this project. The relevant literature that exists covers how to represent programs with graphs [1]; how to detect syntax errors in source code [5]; how to convert code to vectors for neural network processing [6]; and how to carry out tasks on source code using a Gated Graph Sequence Neural Network [7].

3. SPECIFICATIONS, IMPLEMENTATION & SCHEDULE

3.1. IMPLEMENTATION OVERVIEW

The implementation of this project has been divided into three sections. These are: Text, Tree, and Graph. Each section is based on one of the three different neural networks that will be built as part of the project. The Text Section involves implementing the RNN, the Tree Section involves implementing the TBCNN and the Graph Section involves building the Graph-Based neural network.

Each of these sections is further divided into three phases. The first of these is the Implementation Phase and it involves implementing the model. Phase 2 is the Testing Phase where each model is tested and the results of testing are recorded. The third and final phase is the Analysis and Evaluation Phase where the results from testing with each model are evaluated and analysed.

3.2. DATASETS AND DATA COLLECTION

The dataset for this project is divided into two categories: Training Data and Testing Data. The Training Data is used in the Implementation phase to train the models while the Testing Data is used in the Testing Phase. All the datasets are written in the Java programming language [15].

In each category, there are five classes of sorting algorithms: Bubble Sort, Insertion Sort, Merge Sort, Quick Sort and Selection Sort. This data has been collected from various GitHub repositories.

3.3. TECHNICAL SPECIFICATIONS

During each Implementation Phase, I will make use of Supervised Learning (classification) [14] to train each model. The technical specifications of each model are outlined below:

i. The RNN model: This will consist of four layers; the Input Layer, two hidden layers and the Output Layer. The number of neurons in each model will be determined during the implementation process.
ii. The TBCNN model: This will have Input and Output layers, with the number of hidden layers and their neuron counts to be determined during implementation.
iii. The Graph-Based NN model: Similar to the TBCNN model, this model will have an Input and Output Layers with the number of hidden layers and their structures to be determined during implementation.

3.4. PROGRAMMING LANGUAGES AND LIBRARIES

This project will be written using the Python Standard Library [18], TensorFlow [16], Keras [19], Scikit-Learn [17], and the Any Python Tree Library [3] although, there is the possibility that more libraries will be necessary during the different phases of the project.

3.5. IMPLEMENTATION SCHEDULE

As described in section 3.1, the implementation of this project has been divided into three phases. The schedule for implementation is as follows: The Text Section first, where the RNN is built. The Tree Section is next, where the TBCNN is built and finally, the Graph section where the Graph-Based NN is built. Each of these sections is further divided into three phases which will occur in order of: Implementation, Testing and Analysis & Evaluation. The Text Section involves building a conventional RNN NLP model to serve as a foundation to show the shortcomings of NLP models for processing source code.

The Implementation Phase of Tree Section is two-fold. The first stage of this phase is deriving the AST from the program. The second stage of this phase is the implementation of the model based on the AST generated in the first stage.

Similarly, the Implementation Phase of the Graph Section is also divided into two stages. stage 1 involves generating a graph based on the source code, while stage 2 involves building the model based on the graph generated in stage 1.

4. TESTING AND EVALUATION

4.1. TESTING

The Testing Phase is the second phase of the implementation of each model. In this phase, I will make use of 60 different sorting algorithms to test each model. The results of testing each model will be utilized in phase 3 (Analysis and Evaluation) which is described in section 4.2 of this paper.

Based on how efficient the model is in terms of time, there might be an increase in the number of python files used for both training and testing. This will provide a bigger and better picture of how the models really perform.

4.2. EVALUATION AND ANALYSIS OF RESULTS

For each model, this is the final phase of the implementation process. In my evaluation and analysis of the results, I will be focusing on two metrics: Efficiency and Accuracy.

I will be measuring the efficiency of each model by monitoring the amount of time it takes for the model to complete the training process using Python. Each model will be compared to the other two based on the speed at which it completes training. The less time it takes, the more efficient the model

The second metric I will be focusing on is the accuracy of each model. This metric is the percentage of correct classifications made by the model during classification. The higher the accuracy, the more suited the model is for classifying sorting algorithms compared to the other models.

5. POTENTIAL APPLICATIONS AND POSSSIBLE FUTURE WORK

5.1. POSSIBLE FUTURE WORK

There is a wide range of possible future improvements and developments that can be made on this project. One of these is examining how the coding style used to write a program affects the ability of a neural network to understand the code – e.g., code written by the same person vs code written by different people. This involves thinking about how the naming of functions and methods affects program understanding.

Another possible development could be building specific neural networks that process and understand source code without needing to convert to ASTs or graphs. A further development on this could be investigating cross language understanding i.e., examining what happens when training happens in one language and testing happens in another language e.g., training in Python and testing with C.

The outcomes of this project could be expanded even further by building a neural network that can detect errors (syntax and runtime) in programs, by building a NN that takes a program and generates a description of what the program is doing, or, by building a neural network that finds out the time complexity of a program.

5.2. POTENTIAL APPLICATIONS

One potential application of Programming Language Understanding, and this project by extension, is finding errors (syntax and runtime) in programs, specifically larger programs where it is difficult or impractical for programmers to manually search for errors.

Another potential use of this project could be finding out what a program is doing e.g., finding out what type of sorting algorithm a program is implementing.

6. BIBLIOGRAPHY

[1] Allamanis, M., Brockschmidt, M., & Khademi, M. (2017). Learning to represent programs with graphs. *arXiv preprint arXiv:1711.00740*.

[2] Mou, L., Li, G., Jin, Z., Zhang, L., & Wang, T. (2014). TBCNN: A tree-based convolutional neural network for programming language processing. *arXiv preprint arXiv:1409.5718*.

[3] Anytree.readthedocs.io. n.d. Any Python Tree Data — anytree 2.8.0 documentation. [online] Available at: https://anytree.readthedocs.io/en/latest/

[4] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, *20*(1), 61-80.

[5] Tushar Sharma, Vasiliki Efstathiou, Panos Louridas, and Diomidis Spinellis. 2019. On the Feasibility of Transfer-learning Code Smells using Deep Learning. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (May 2019).

[6] Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, *3*(POPL), 1-29.

[7] Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

[8] Chowdhary, K. (2020). Natural language processing. *Fundamentals of artificial intelligence*, 603-649.

[9] Agnarsson, G., & Greenlaw, R. (2006). *Graph theory: Modeling, applications, and algorithms*. Prentice-Hall, Inc.

[10] Medsker, L. R., & Jain, L. C. (2001). Recurrent neural networks. *Design and Applications*, *5*, 64-67.

[11] Kundel, D., 2020. *Introduction to Abstract Syntax Trees*. [online] Twilio Blog. Available at: https://www.twilio.com/blog/abstract-syntax-trees.

[12] Wang, S. C. (2003). Artificial neural network. In *Interdisciplinary computing in java programming* (pp. 81-100). Springer, Boston, MA.

[13] En.wikipedia.org. n.d. *Tree (data structure) - Wikipedia*. [online] Available at: https://en.wikipedia.org/wiki/Tree_(data_structure)

[14] IBM Education., 2020. *What is Supervised Learning?*. [online] Ibm.com. Available at: https://www.ibm.com/cloud/learn/supervised-learning

[15] Docs.oracle.com. n.d. *Java Programming Language*. [online] Available at: https://docs.oracle.com/javase/7/docs/technotes/guides/language/

[16] TensorFlow. n.d. *TensorFlow* . https://www.tensorflow.org.

[17] Scikit-learn.org. n.d. *scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation*. [online] Available at: https://scikit-learn.org/stable/

[18] Docs.python.org. n.d. *The Python Standard Library — Python 3.10.4 documentation*. [online] Available at: https://docs.python.org/3/library/

[19] Keras Team., n.d. *Keras: The Python deep learning API.* https://keras.io