**COMPARATIVE CASE STUDY BETWEEN GATED GRAPH NEURAL NETWORKS**

**VERSUS**

**RELATIONAL GRAPH CONVOLUTIONAL NETWORKS FOR THE VARIABLE MISUSE TASK**

**USING PYTHON AND DEEP LEARNING**

**By**

**Mukund Raghav Sharma**

A Capstone Project Paper Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in

Data Science

University of Wisconsin – La Crosse

La Crosse, Wisconsin

May 2020

**DEDICATION**

**To my grandparents and my favorite planet, Saturn.**

ABSTRACT

COMPARATIVE CASE STUDY BETWEEN GATED GRAPH NEURAL NETWORKS

VERSUS

RELATIONAL GRAPH CONVOLUTIONAL NETWORKS FOR THE VARIABLE MISUSE TASK

USING PYTHON AND DEEP LEARNING

MUKUND RAGHAV SHARMA

MASTER OF SCIENCE IN DATA SCIENCE

UNIVERSITY OF WISCONSIN – LA CROSSE, 2020

Identifying bugs in source code has been an extremely important part of software development since the inception of the industry. The majority of static analysis, the analysis of software without actually executing programs, is rule based without much involvement of deep learning until fairly recently.

This paper engages in a comparative study of determining the more performant graph neural network model on the basis of test accuracy between Gated Graph Neural Network (GGNN) models and Relational Graph Convolutional Network (RGCN) models on the Variable Misuse Task, a prediction task involving choosing the correct variable based on all the variables of the same type in a particular scope.

The data is of source code from the files of 25 trending C# repositories that are converted into a modified Abstract Syntax Tree to represent a directed graph whose vertices that represent the tokens and relationships between the tokens are represented by edges. Each of these vertices are associated with one of the aforementioned type of networks for the training phase after a particular embedding is computed for each token.

The comparison to decide the more efficient model is based on the test accuracy of all the repositories, an esoteric repository and an extremely popular repository to cover the spectrum of different types of repositories. The results show that the RGCN based models outperformed the GGNN models for all cases, albeit, within < 5% range.

*Keywords: Deep Learning, Graph Neural Networks, Tensorflow, Sequence Models, Convolutional Models, Learning from Code, Static Analysis.*

**TABLE OF CONTENTS**

DEDICATION……………………………………………………………………………iii

ABSTRACT………………………………………………………………………………iv

TABLE OF CONTENTS………………………………………………………………….v

LIST OF FIGURES……………………………………………………………………...vii

CHAPTER 1 – INTRODUCTION………………………………………………………..1

Project Background………………………………………………………………………..1

Objectives…………………………………………………………………………………3

Rationale and Inspiration for Undertaking the Project……………………………………4

CHAPTER 2 – LITERATURE OVERVIEW………………………….………………….6

Previous Work………………………….………………………….………………………6

Other Presentations………………………….………………………….………………..10

CHAPTER 3 – PROJECT OVERVIEW…………………..…………………..…………11

General Approach / Layout…………………..…………………..………………………11

CHAPTER 4 – GRAPH NEURAL NETWORK 101……………………………………13

Goals……………………………………………..………………………………………13

The Var Misuse Task……………………………………………..………………………14

Graph Neural Networks………………………………………………………………….17

Gated Graph Neural Networks…………………………………………………………...27

Relational Graph Convolutional Neural Networks………………………………………29

CHAPTER 5: PRELIMINARY STEPS IN CASE STUDY……………………………..31

Problem Definition………………….………………….………………….……………..31

Interview Notes………………….………………….………………….………………...32

CHAPTER 6: SOURCE DATA EXPLAINED………………….………………….……34

Introduction………………….………………….………………….…………………….34

Tokens………………….………………….………………….………………….………40

Type Hierarchy………………….………………….………………….…………………40

Structure of the Typical Graph File ………………….………………….……………….41

Other Data Sources………………….………………….………………….…………….42

CHAPTER 7: METHODOLOGY AND RESULTS………………….………………….44

Introduction………………….………………….………………….…………………….44

Experimental Design………………….………………….………………………………45

Details Related to the Pipeline………………….………………….…………………….50

Conducting the Experiments………………….………………….………………………59

Results………………….………………….………………….………………………….64

Next Possible Steps………………….………………….………………….…………….65

CHAPTER 8: CONCLUSION………………….………………….……………………67

REFERENCES………………….………………….………………….………………...69

**LIST OF FIGURES**

Fig. 1: Timeline of Graph Neural Networks………………………………………………6

Fig. 2: Example 1 of Application of the VarMisuse Task in C#………………………...14

Fig. 3: Example 2 of Application of the VarMisuse Task in C#…………………………16

Fig. 4: An Example of a Graph…………..…………..…………..…………..…………..18

Fig. 5: Simple Abstract Syntax Tree…………..…………..…………..…………..…….20

Fig. 6: Modified Abstract Syntax Tree that Represent Programs…………..……………21

Fig. 7: Graph Structure and Messages For Each Node…………..………………………24

Fig. 8: Message Passing And Aggregation by Nodes via a Recurrent Unit……………..25

Fig. 9: Unrolled Effect Of the Graph Where Nodes Become Positionally Aware ………26

Fig. 10: Formula for the GGNN State Update Per Node…………..…………………….28

Fig. 11: Formula for the RGCN Node State Update Per Node…………..………………29

Fig. 12: Data Transformation Pipeline…………..…………..…………..……………….35

Fig 13: Example Representation of the Next Token Of the Directed Graph…………….35

Fig. 14: Representing the LastWrite in the Source Code As a Graph…………………..36

Fig. 15: Listing of all the Directories From the Graph Data Set………………………...38

Fig. 16: Typical Directory Structure of Data From A Repository………………………..39

Fig. 17: Example of a Record of the RepositoryInfo…………………………………….43

Fig. 18: All Repositories Considered…………..………………………………………...46

Fig. 19: CommonMark.NET Details…………..………………………………………...48

Fig. 20: Dapper Details…………..…………..…………………………………………..49

Fig. 21: List of Models…………..……………………………………………………….52

Fig. 21: List of Gated Graph Neural Networks…………..………………………………53

Fig. 23: List of Tasks And Hyperparameters…………..…………..…………..………...54

Fig. 24: List of Scripts…………..…………..…………..……………………………….55

Fig. 25: List of Utilities…………..…………..…………..………………………………56

Fig. 26: Mockup of the Pipeline…………..…………..…………………………………58

Fig. 27: Data Needed For Each of the Experiments…………..…………………………59

Fig. 28: Training Command Line Args…………..…………..…………………………..60

Fig. 29: Command Line Output For Training…………..…………..……………………61

Fig. 30: Testing Command Line Arguments…………..…………..……………………..62

Fig. 31: Testing Terminal Output…………..…………..………………………………...63

Fig. 32: Final Results…………..…………..…………..………………………………...64