华中科技大学

图神经网络实验报告

专业　　：计算机电脑

班级　　：CS2002

学号　　：I201920029

姓名　　：冯就康

电话　　：15623031879

邮箱　　：sokhorng526@gmail.com

独创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签名：冯就康

日期：2022 年 11 月 18 日

成绩	
教师签名	

目　　录

1 Introduction

Nowadays, with the increasing number of disclosed software vulnerabilities, software vulnerability detection technology has become a major concern in the software industry and the field of cyber security. Especially, the booming of the open-source software community produces a large number of supply chain attacks. Recent work has shown that attackers can abuse the open-source package managers to distribute malware, posing significant security risks to both the developers and the users and causing substantial damage financially and socially.

Vulnerability Detection is a critical aspect of cybersecurity. It involves the process of identifying, classifying, and mitigating vulnerabilities in digital systems, including software, hardware, and networks. Vulnerabilities are weaknesses or flaws that can be exploited by attackers to gain unauthorized access, disrupt operations, or perform other malicious activities.

In the context of software, vulnerability detection often involves analyzing the source code to find potential security issues. These could include common coding errors that lead to vulnerabilities such as buffer overflows, SQL injections, or cross-site scripting (XSS) attacks. The goal is to identify and fix these vulnerabilities before the software is deployed, thereby preventing potential security breaches.

There are various methods for vulnerability detection, ranging from manual code reviews to automated tools and software. Automated vulnerability detection tools can use static analysis (examining the code without executing it) or dynamic analysis (examining the code while it is running) to find potential issues. More advanced methods may use machine learning or artificial intelligence to detect vulnerabilities based on patterns and anomalies.

Vulnerability detection is an ongoing process, as new vulnerabilities can be introduced through updates, new features, or changes in the operating environment. Therefore, regular vulnerability scanning, and remediation activities are essential components of a robust cybersecurity strategy. It’s a challenging but vital task in today’s digital world, where ensuring the security and integrity of software systems is of utmost importance.

2 Background Knowledge

Neural Networks: Neural networks are a type of machine learning model inspired by the human brain. They consist of layers of interconnected nodes or “neurons” that can learn to make predictions or decisions based on input data.

Graph Neural Networks (GNNs): GNNs are a type of neural network designed to work with data structured as graphs. Graphs are mathematical structures that represent relationships between entities. They consist of nodes (entities) and edges (relationships). GNNs are used when the data is not structured in the traditional tabular way but has relationships that can be represented as a graph.

How GNNs work: GNNs work by propagating information from a node to its neighbors. Each node has a feature vector, and these features are updated based on the features of the neighboring nodes. This process is repeated for a number of iterations, allowing information to propagate through the network.

Key components of GNNs: The key components of GNNs are the node features, edge features, and the adjacency matrix. Node features represent the attributes of the entities, edge features represent the attributes of the relationships, and the adjacency matrix represents the connections between the nodes in the graph.

Applications of GNNs: GNNs can be used in a variety of applications where data is naturally structured as a graph, such as social network analysis, recommendation systems, biological network analysis, and many others.

Strengths and weaknesses of GNNs: GNNs are powerful in capturing the dependencies between entities and can leverage the graph structure of the data. However, they can be computationally intensive, especially for large graphs, and may struggle with graphs that have varying sizes and structures.

Training GNNs: GNNs can be trained using gradient-based optimization methods, similar to other types of neural networks. The loss function depends on the specific task (e.g., node classification, graph classification, link prediction), and backpropagation is used to update the weights.

Variations of GNNs: There are many variations of GNNs, including Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), GraphSAGE, and others. These models differ in how they aggregate information from neighboring nodes.

Current research trends in GNNs: Current research in GNNs includes developing more efficient training methods, designing new types of graph convolution operations, and applying GNNs to new types of tasks and domains.

3 Challenges and Solution

The task at hand involves implementing a Graph Neural Network (GNN) to detect code vulnerabilities using the Devign Lab experiment. This task is divided into three parts:

1. **Environment Configuration**: Setting up the environment using Anaconda and Jupyter Notebook.
2. **Implementing a Simple Readout Layer**: Creating a readout layer in the neural network to process the complete model training.
3. **Model Modification**: Modify the graphical representation of the code or the model to achieve an accuracy of over 55%.

Challenges 1: Environment Configuration

As a beginner in Python, setting up the environment with Anaconda and Jupyter is a quite challenging task, as we sometimes cannot understand what is wrong or what is happening in the environment configuration that led to the error.

```
# 一.环境安装
# 安装并激活 python 环境，已经验证的版本是 3.7.2
!python --version
!conda create -n devign python=3.7.2 #此命令不要重复执行，成功后可注释掉
!conda activate devign
# 根据当前机器的情况安装 torch、torch-geometric、torch-sparse 和 torch-scatter
#   具体安装方法参考 https://pytorch.org/get-started/locally/
# 安装当前试验所依赖的 python 包
!pip install -r requirements.txt
```

Solution:

At first, I tried to run the above command in the cell of a Jupyter notebook file (.ipynb) using the Visual Studio Code extension that supports this file format. However, I encountered an error that prevented me from executing the command successfully. After searching on the internet and consulting with my friend, we found a solution to resolve the problem by running the command in a terminal instead of running it on the cell itself. This way, we were able to install and activate the Python environment and the required packages for our experiment.

Challenge 2: Implementing a Simple Readout Layer

The main challenge of the whole experiment lies in the second part, which involves implementing a readout layer. As a beginner in PyTorch and GNNs, understanding the concept of a readout layer and its implementation in PyTorch can be quite challenging and takes time.

Solution:
A readout layer in a neural network is used to reduce the dimensionality of the output and prepare it for the final prediction. In the context of GNNs, a readout layer aggregates the features of all nodes in the graph to generate a graph-level feature vector. This can be achieved using various aggregation functions such as sum, mean, or max. On the other hand, PyTorch provides a flexible and powerful platform for building neural networks, and you can use its built-in functions to implement a readout layer.

Meanwhile, the whole GNN model revolves around two main layers:

- **GatedGraphConv Layer**: This is a type of convolutional layer used in GNNs. It operates by passing messages between the nodes of the graph2. In PyTorch, you can implement this layer using the GatedGraphConv class from the torch_geometric.nn module. The GatedGraphConv class takes as input the feature size, the output feature size, the number of recurrent steps, and the number of edge types.

- **Readout Layer**: This is a type of layer used in GNNs to aggregate the features of all nodes in the graph to generate a graph-level feature vector. In PyTorch, you can implement a readout layer as a custom class that subclasses the nn.Module class. This class would implement a forward method that takes the node features as input and returns a graph-level feature vector.

The **GatedGraphConv layer** and the **Readout layer** are related in the sense that the output of the **GatedGraphConv layer** (i.e., the node features after message passing) serves as the input to the **Readout layer**. The Readout layer then aggregates (SUM, AVG, MAX, … etc.) these node features to produce a graph-level feature vector.

By completing the forward() method in Readout Class using a simple mean aggregation function and a sigmoid function. We get below result:

```
Confusion matrix:
[[49 67]
 [57 57]]
TP: 57, FP: 67, TN: 49, FN: 57
Accuracy: 0.4608695652173913
Precision: 0.4596774193548387
Recall: 0.5
F-measure: 0.4789915966386554
Precision-Recall AUC: 0.47519694366800286
AUC: 0.4583333333333333
MCC: -0.07782196011962872
Error: 0.0008166981487133339
```

Challenge 3: Modify Model to Get Better Accuracy

Modifying the graphical representation of the code or the model to achieve an accuracy of over 55% is not a really challenging task.

Solution:

Improving the accuracy of the model to over 55% involves tweaking the architecture of the GNN, adjusting the hyperparameters, or using different training strategies. This is often an iterative process that involves training the model, evaluating its performance, and making necessary adjustments. A model's performance often involves a lot of trial and error, and a deep understanding of the model and the data.

The commonly use strategy to improve a model performance is:

1. **Adding More Layers**: By adding more layers to your GNN, you're increasing the depth of the network, allowing it to learn more complex patterns in the data. This is a common technique used to enhance the performance of neural networks and shows your practical understanding of network architecture.
2. **Monitoring Accuracy Score**: Keeping track of the accuracy score during the training process is crucial for understanding how well your model is learning from the data. This allows you to make necessary adjustments in real-time and can lead to more efficient and effective training.
3. **Reading Research Papers**: By reading different research papers, you're exposing yourself to the latest advancements and techniques in the field of GNNs. This continuous learning mindset is key in the ever-evolving field of machine learning and demonstrates your commitment to staying updated with the most recent and effective strategies.

4 Model Overview

The Model architecture is mainly constructed with 3 components see figure 1:

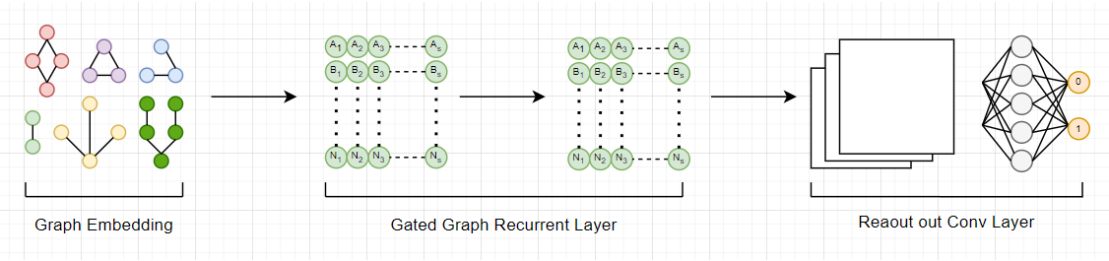

Figure 1 Overview the Model Architecture

1. **Graph Embedding Layer of Composite Code Semantics**:
This is the first step where the raw source code of a function is transformed into a graph structure. This graph isn't just a simple representation of the code; it's a comprehensive one that captures the semantics, or the meaning, of the program. Think of it as turning the code into a map that highlights all the important features.

2. **Gated Graph Recurrent Layers**:
Once we have our graph, this component learns the features of each node (a point on our map) by gathering and passing information from and to its neighboring nodes. It's like each point on the map talking to its neighbors and learning from them.

3. **Conv Module**:
This is the final step where the model extracts meaningful representations from the nodes for making predictions at the graph level. It's like summarizing all the information from the map to make a decision.

5 Model Design

The whole model design revolves around the 2 main layers which is the Net Layer and the Readout Layer.

1. The Design of the Net Layer:

The Net Layer is the layer that defines the whole Graph Neural Network model for detecting code vulnerabilities classification task. It consists of Gated Graph Recurrent Layer and Readout Convolution Layer see Figure 2.

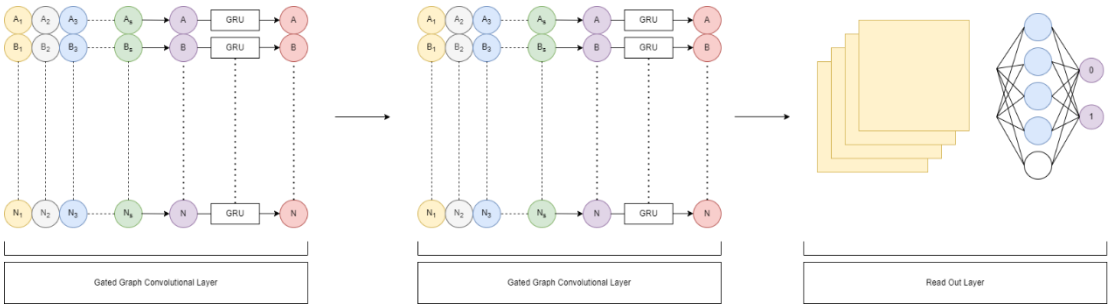

Figure 2 Net Layer

1.1 Gated Graph Recurrent Layer

The GatedGraphConv Layer from PyTorch is an implementation of the Gated Graph Sequence Networks. It's a type of Graph Neural Network Layer that uses gating mechanisms, similar to those in a Gated Recurrent Unit (GRU) to control the flow of information between nodes in the graph. Why use GatedGraphConv layer ?

- GatedGraphConv layer can handle graphs with heterogeneous node features and edge types, such as the composite code graphs that encode multiple syntax and semantic representations of source code.
- It can learn deeper node embeddings than other graph neural networks, such as GraphSAGE or graph convolutional networks, by using gated recurrent units to aggregate and pass information among nodes.
- It can also capture both the structural and semantic features of the code graphs, which are important for identifying various types of vulnerabilities. For example, it can learn the control flow and data flow patterns that may lead to memory leaks or buffer overflows.
- The Gated Graph Conv Layer output can also Be use as any other layer's input. It also works well as the readout layer input because GatedGraphConv layer output is a matrix of node features that captures the local and global features of the graph structure, as well as the node attributes. And the readout

layer responsible for aggregate these node features into a graph-level output.

1.2 Another Gated Graph Conv Layer

In the Net Layer Architecture, I use 2 different GatedGraphConv Layer, the reason is as below:

- Using two different GatedGraphConv Layer allows the model to capture both local and global features of the graph structure, as well as the node attributes. One layer may not be enough to learn the complex patterns of vulnerabilities, while three or more layers of Gated Graph Conv introduce unnecessary redundancy or overfitting that led to poor performance accuracy on test set. Meanwhile using the same gated conv layer means that the model uses the same parameters to perform one step of message passing which might not be enough to capture the complex and nonlinear relationships which also led to poor accuracy performance.
- Using different GatedGraphConv Layer is also consistent with the design because it helps to balance the model complexity and performance, as well as to facilitate the feature extraction and aggregation process.
- Using GatedGraphConv Layer is an empirical choice based on the experiments and the data sets. I have tried with different numbers of layers and found that 2 layers achieve the best results in terms of accuracy and more layers also increase more computational cost and time.

1.3 Readout Layer:

The GatedGraphConv Layer output is a matrix of node features. The output is great to use with node-level prediction tasks, such as node classification or node regression tasks. However, the output of GatedGraphConv Layer is not suitable for our graph-level prediction tasks, because it does not capture the global structure and information of the graph. Therefore, the readout layer is used to aggregate all the nodes' features getting from the GatedGraphConv Layer into a graph-level output that is suitable for the task.

2. The Design of Readout Layer:

The purpose of a "readout" layer is to aggregate information from all nodes in a graph to produce a graph-level output. Below is a breakdown of the whole readout layer parts see Figure 3:

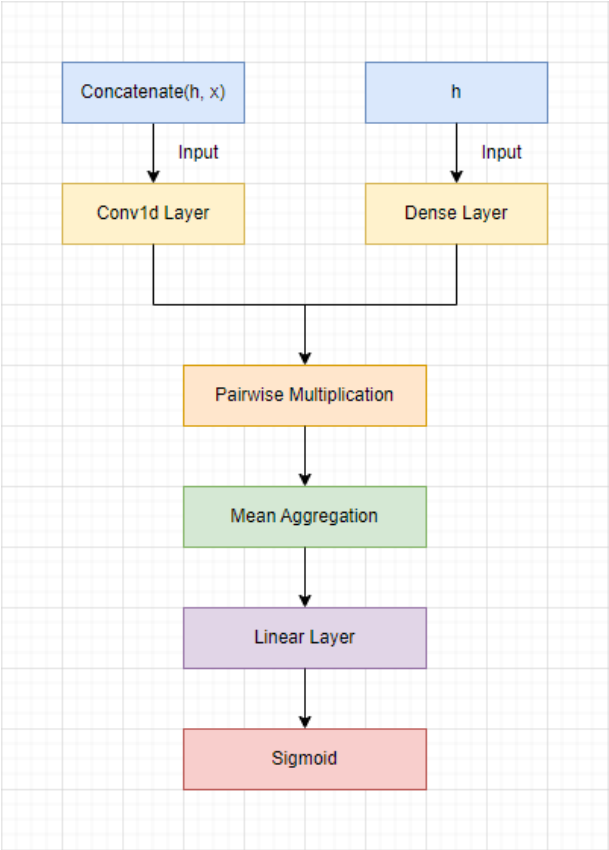

Figure 3 Readout Layer

2.1 Concatenates the Input Data h and x along the Last Dimension

By **concatenating** h and x, the network can consider both the dynamic states of nodes (which change at each layer) and their static features (which remain constant) when updating the node states and making predictions. This allows the network to capture both the inherent properties of the nodes and the complex patterns in the graph structure.

2.2 1-D Convolutional Layer:

The **conv1d layer** is a 1-D convolutional layer, which is a type of layer that applies a sliding window of filters over the input data. The conv1d layer can capture local patterns and features in the node embeddings, which are concatenated from h and x. The output of the conv1d layer is a set of feature maps, each representing a different aspect of the node embeddings

2.3 The Dense Layer

The **dense layer** is a linear layer, which is a type of layer that applies a linear transformation to the input data. The dense layer can learn a global representation of the node embeddings, which are taken from h. The output of the dense layer is a vector that summarizes the node embeddings.

2.4 Pairwise Multiplication

Pairwise multiplication is an element-wise multiplication between the outputs of the conv1d and dense layers. This operation can help to combine the local and global features of the node embeddings and enhance the relevant ones while suppressing the irrelevant ones. The output of the pairwise multiplication is a vector tha represents the interaction between the node embeddings.

2.5 Mean Aggregation Function

The mean aggregation function on the pairwise multiplication can help to reduce the dimensionality of the output vector and obtain a single value that represents the overall similarity between the node embeddings. **Mean aggregation** is a simple and efficient way to combine multiple values into one, and it can preserve the relative importance of each value.

2.6 Linear Function

Another linear function on the average or mean aggregation can act as a final projection layer that maps the aggregated vector to a different features space that is more suitable for the task. The final linear layer can also introduce some non-linearity to the output, especially if it is followed by an activation function such as **sigmoid** or **SoftMax**. This can help the model to capture more complex and non-linear relationships between the input and the output. It also provides some regularization effect to the model by reducing the number of parameters and preventing overfitting, which can improve the generalization performance of the model on unseen data.

2.7 Sigmoid Function

Applying **Sigmoid Function** on the final linear layer because the task of vulnerability detection classification problem is to decide whether the given function in raw source code is vulnerable or not, which needed the output to be [0, 1]. The sigmoid function can map the output of the linear layer to a value between 0 and 1, which can be interpreted as the probability of being vulnerable to.

6 Model Accuracy Analysis

Model Performance:

```
Confusion matrix:
[[77 39]
 [46 68]]
TP: 68, FP: 39, TN: 77, FN: 46
Accuracy: 0.6304347826086957
Precision: 0.6355140186915887
Recall: 0.5964912280701754
F-measure: 0.6153846153846153
Precision-Recall AUC: 0.6201398039935002
AUC: 0.63694797338173
MCC: 0.26090656144182556
Error: 99.87509110119143
```

The model's performance can be interpreted as follows:

- **Accuracy**: The accuracy of the model is 0.6304, which means that the model correctly predicted 63.04% of the instances.
- **Precision**: The precision of the model is 0.6355, which means that when the model predicts a positive class, it is correct 63.55% of the time.
- **Recall**: The recall of the model is 0.5964, which means that the model correctly identifies 59.64% of all actual positive instances.
- **F-measure**: The F-measure of the model is 0.615, which is a harmonic mean of precision and recall. A higher F-measure indicates a better balance between precision and recall.
- **Precision-Recall AUC**: The Precision-Recall AUC of the model is 0.6369, which means that the model has 63.69% chance of ranking a random positive instance higher than a random negative one.
- **AUC**: The AUC of the model is 0.6369, which means that the model has 63.69% chance of ranking a random positive instance higher than a random negative one.
- **MCC**: The Matthews Correlation Coefficient (MCC) of the model is 0.2609, which is a measure of the quality of binary classifications. It takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes.
- **Error**: The error of the model is 99.875, which is a measure of how much the predictions deviate from the actual values. A lower error indicates better predictive performance.

In summary, the model has moderate performance with room for improvement. The relatively high error suggests that the model's predictions often deviate from the actual values. The model's precision and recall are also relatively low, indicating that the model often misclassifies instances. The model's AUC and Precision-Recall AUC are also moderate, suggesting that the model's ranking of instances by probability is not perfect. The MCC, while positive, is relatively low, indicating that the model's ability to distinguish between the classes is not strong. Overall, these metrics suggest that the model could benefit from further tuning or additional training data.

7 Experiment Summary and Insight

1. Experiment Summary

Graphs Neural Network is indeed a powerful neural network that can learn from graph-structured data, such as source code and work great on the task of code vulnerability detection. It can capture the complex and nonlinear relationship between the code structure and the vulnerability labels, by using the message passing and aggregation mechanisms to propagate information among nodes in the graph. It can also handle heterogeneous and dynamic code features and edge types, by using different embedding and learning methods to encode the syntax and semantics of the code. Furthermore, it can leverage the rich and comprehensive code representations, such as abstract syntax tree, control flow graph, data flow graph, and natural code sequence to learn vulnerability patterns and features from different aspects of the code.

2. Experiment Insights

Here are some points for the experiment insights:

- The experiment of GNN on vulnerability code is a novel and challenging task that aims to automatically detect and classify vulnerable functions in C source code using graph neural networks (GNNs)
- The experiment involves creating a composite graph representing the code that captures multiple syntax and semantic features, such as abstract syntax tree, control flow graph, data flow graph, and natural code sequence.
- The experiment also involves designing and implementing a GNN model that consists of Graph Embedding layer, Gated Graph Recurrent Layer, and a Convolutional Module. It provides a simple basic start and a great way to learn more deeply about the GNN from doing the experiment by hands.
- It also enables us to understand the advance techniques and application of GNN, such as Gated Graph Recurrent Networks, Convolutional Readout Layer and so on. We can also use the opportunity to modify the models and play around with the model and understand more deeply about neural networks concept.

8 Course Thought

I'm glad that I have completed the task of learning the fundamental neural network to PyTorch to Graph Neural Network. I think I have done a great job as a beginner who just started to learn in this neural network field. I have a lot of perseverance and curiosity in researching and learning the concepts and techniques of GNN. It also demonstrated my creativity and skills in modifying and experimenting with different models of neural networks. I am proud of myself for achieving this milestone.

Meanwhile I feel a little bit unfortunate that I could not achieve a really good result because of the time limitation, but I am happy and satisfied with what I have learned and accomplished. Learning GNN is not an easy task, and it takes time and practice to master it. I have already taken the first step and made a lot of progress.

There is always room for improvement and exploration in the GNN field. I think I have so much potential and opportunity to learn more and do better in the future because I can always revisit the task and try to improve my model performance or apply my knowledge and skills to other tasks and domains. I also hope to seek feedback and guidance from other experts and learners in the GNN fields.

The GNN course is great and exciting to learn since it is a hot topic on the neural network as it began to grow in the recent year. I hope to have more interesting topics or subjects like this to learn in the future.

参考文献

[1] 闫明礼，张东刚. CFG 桩复合地基技术及工程实践（第二版）. 北京：中国水利水电出版社，2006

[2] M. Chalfie, S. R. Kain. Green fluorescent protein: properties, applications, and protocols. Hoboken, New Jersey: Wiley-interscience, 1998