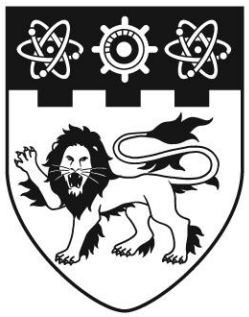

NANYANG TECHNOLOGICAL UNIVERSITY

Universal Adversarial Attacks on Graph Neural Networks

Submitted by: Liao Chang
Matriculation Number: U1722904L

Supervisor: A.P Tay Wee Peng
Co-supervisor: Dr. Song Yang

School of Electrical & Electronic Engineering

A final year project report presented to the Nanyang Technological University in partial fulfilment of the requirements of the degree of Bachelor of Engineering

2021

Table of Contents

Abstract

This project aims to study the robustness of graph-level graph neural networks (GNNs) against universal adversarial attacks in white-box and grey-box scenarios. Graph-level GNNs are widely used in critical domains such as quantum chemistry, drug discovery, while the robustness of them against universal adversarial attacks is unknown. In particular, there is currently no universal adversarial attack algorithm applicable to graph-level GNNs. In this project, a data-free universal adversarial attack for graph-level GNNs is proposed. We conducted several experiments to show that:

1. Graph-level GNNs are not robust against universal adversarial attacks even under grey-box scenarios.
2. Node features can be used as node embeddings with better performance than using the difference between graphs as node embedding.
3. Betweenness centrality may better reflect the importance of a node in an adversarial attack context.

Acknowledgements

Firstly, I would like to express my very great appreciation to my family and friends for supporting me in pursuing higher-level education overseas physically and mentally. It has been a challenging journey studying alone overseas. Their support and encouragement are always joyful to have throughout these years.

Secondly, I would like to express my deep gratitude to my FYP supervisor Prof. Tay Wee Peng, for the patient guidance, valuable critiques, and pieces of expert advice for this research work via regular meetings. This project was stuck in surrogate data generation, and Prof. Tay pointed out this problem during one of the regular meetings so that I was able to catch up with the progress.

Finally, I would like to thank Dr. Song Yang, a research fellow from Nanyang Technological University, for his patience and guidance throughout this project. Dr Song Yang proposes many brilliant ideas, and I have learned a lot from him through our discussions.

Acronyms

GNN	Graph Neural Network
CNN	Convolutional Neural Network
GCN	Graph Convolution Network
UAA	Universal Adversarial Attack
GUAA	Graph Universal Adversarial Attack
HGP-SL	Hierarchical Graph Pooling with Structure Learning
AAA	Ask, Acquire and Attack

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Graph structure has been adapted to represent real-world data for centuries. In recent years, neural networks have been deployed on graph-structured data and played essential roles in various domains, including social networks [1, 2], medicine discovery [3, 4], and intelligent transport systems [5, 6]. However, as a common flaw of learning-based algorithms, graph neural network (GNN) is also shown to be vulnerable under adversarial attacks [7, 8], which craft malicious examples to mislead the victim GNN to an erroneous output. Based on the victim model's scale, graph adversarial attacks can be categorized into node classification attacks and graph classification attacks. Meanwhile, graph adversarial attacks can also be classified by the target of perturbation as a graph that consists of nodes and edges. Any attack that only perturbs the node features is a node attack [9, 10], while attacks on graph structures (edges) are called topological attacks [11, 12].

Since the discovery of graph adversarial attacks in 2018 by Zügner et al. [8], continuous efforts have been putting into studying the robustness of graph neural networks against them [13, 14]. Previous research showed it is possible to generate a single perturbation for a dataset and apply it to all samples in that dataset leading to a high overall misclassification rate [15-23]. These adversarial attacks are called universal adversarial attacks (UAA). Within those UAA algorithms, there are many of them targeting images [16-20], texts [15], and other unstructured data [21], but only a few aim on graph-structured data [22, 23]. In fact, [22, 23] are the only two works in this specific field. Furthermore, both are limited to GNNs on node classification tasks. In other

words, the robustness of GNNs on graph classification tasks in some critical applications such as protein characteristics predictions towards GUAA remains unknown.

Previous GUAAs assume the adversary has full knowledge about the victim model (white-box scenario) [16, 17], including training data, model structure, and weights, while in practice, training data of a GNN is highly valuable and less likely to leak to the adversary. Intending to understand the robustness of learning-based models under a grey-box scenario, [19] proposed to generate surrogate data (called class impression) based on gradients of the objective function with respect to randomized input, while [17, 18] proposed to utilize hidden features produced by interior layers of the victim model. However, these methods are all on images. Due to graph-structured data's topological nature, those attack algorithms fail to construct an adversarial perturbation with topological structure and hence are not applicable for graphs. Hence, the robustness of GNN models under the grey-box scenario also remains unknown.

1.2 Objective and Scope

This project aims to study the robustness of graph-level GNNs against universal adversarial attacks under a grey-box scenario where the victim model's training data is unknown to the adversary.

To tackle the problem mentioned earlier, we propose a universal attack algorithm and a surrogate data generation algorithm for graph-level GNNs. Surrogate data are for the proposed universal adversarial attack algorithm to generate perturbations on, hence there is no requirement for the adversary on training data. Also, as the victim model in this project is a model trained on PROTEINS dataset, and the secondary structure elements (SSEs) of proteins are widely known, it is reasonable to assume that the adversary knows all

possible SSEs (nodes). Hence, the problem becomes: proposing an algorithm to select nodes from a candidate pool such that appending these selected nodes to benign graphs lowers down the performance of the victim model.

The scope of this project is listed as follows to organize it clearly:

1. Graph-level neural networks as victim models
2. Robustness against universal adversarial attack
3. A grey-box setting where actual training data is unknown and SSEs are known to the adversary

The contributions of this project are summarized as follows:

1. We proposed a universal adversarial attack against graph-level GNNs, which confirms the vulnerability of these GNNs. To the best of our knowledge, our research is the first work on graph-level GUAA.
2. We proposed a surrogate data generation method for graph-structured data while only node feature distribution is known to the adversary. Adversarial perturbations on surrogate data have shown high aggression against victim GNN.

1.3 Organizations

This report is organized into five chapters. Chapter 2 will be on the literature review of research works related to this project. Chapter 3 will introduce the proposed algorithms and methodology. Chapter 4 will discuss the experiments' design with explanations for their corresponding functions, present and analyse experimental results. Chapter 5 will be the overall conclusion and recommended future works for the project.

Chapter 2

Literature Review

This chapter will introduce the history of machine learning models on graph-structured data, the history of adversarial machine learning, attack methodologies for graph adversarial attacks, existing universal adversarial attacks, and surrogate data generation algorithms.

2.1 History of Graph Neural Network

2.1.1 Graph-structured data

2.1.1.1 Definition

A graph $G = (V, E)$ is a data structure that consists of a set of vertexes (nodes) V and a set of edges E. E is commonly represented in a matrix form called adjacency matrix (A) and vertexes are also represented in a matrix form X containing vertex features, hence the graph $G = (V, E)$ can also be annotated as $G = (X, A)$. Based on edges' attribute, graphs can be weighted (with edge attributes) or unweighted (no edge attributes). Also, based on the symmetry of the adjacency matrix, graphs can be directed (there is an edge $E_{i,j}$ from node i to node j but there is no edge connecting node j to node i, i.e., $A_{i,j} = 1, A_{j,i} = 0$) or undirected ($A_{i,j} = A_{j,i} \forall i, j \in n$).

2.1.1.2 Vertex Centrality

The centrality of a vertex identifies the importance of this vertex within a graph. There are many implementations for vertex centrality, Degree Centrality (deg), Betweenness Centrality (btw) and Eigenvector Centrality (eig) will be applied in the following experiments, hence they will be covered

in this section.

Degree Centrality is the first and most conceptually straightforward implementation. The degree of a vertex is the number of edges that the node has. Degree centrality of a vertex is the degree of this vertex, i.e., $C_D(v) = \deg(v)$.

Betweenness Centrality qualifies the number of times a node in the shortest path between two other nodes. To compute the betweenness centrality C_B, the shortest paths between every pair of nodes are computed, then the fraction of those shortest paths passing node V over the number of all shortest paths is defined as the betweenness centrality. More compactly, it is defined as

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths (number of node pairs) and $\sigma_{st}(v)$ is the number of those paths that pass through v. It is defined as the summation form as there might be several shortest paths from node s to node t.

Eigenvector Centrality measures the influence of a node in a graph. It assigns relative scores to all the nodes in a graph, in that nodes connected to high-score nodes will also have higher scores and vice versa. Eigenvector Centrality is also the conceptual origin of Google's famous PageRank algorithm. The formula to calculate the Eigenvector centrality C_E is defined as

$$C_E(v) = \frac{1}{\lambda} \sum_{t \in \mathcal{N}(v)} C_E(t) = \frac{1}{\lambda} \sum_{t \in G} A_{v,t} C_E(t)$$

Annotating $C_E(v)$ as x, and the centrality scores of all nodes as X, it can be simplified to

$$AX = \lambda X$$

which is the eigenvalue equation. As all the elements in the adjacency matrix are non-negative, by the Perron-Frobenius theorem [24], there is a unique largest real and positive eigenvalue. In the end, it outputs the desired centrality scores X.

2.1.2 Graph and Embedding

Embedding is a standard method in Natural Language Processing (NLP) and other fields requiring representation learning for discrete tokens. It casts input low-dimension data to a higher dimension resulting in much more information represented and revealing more hidden information compared to traditional manual feature engineering. Graph embedding learns to present graph nodes, edges in vectors. DeepWalk [25] is regarded as the first graph embedding method applying SkipGram model on the generated random walks. Following breakthroughs made by similar approaches such as node2vec [26], LINE [27], and TADW [28] fail to overcome two severe drawbacks due to the nature of embedding. Firstly, graph embedding's computational cost grows linearly with the number of nodes since no parameter is shared in the embedding encoder for different nodes. Secondly, unlike the public common pre-trained word embedding encoder for texts build on an enormous corpus with 2,500 million words[29], graphs are built for different purposes. Hence, there is no common large-scale dataset for graphs, and graph embedding methods are also unable to generalize because of this.

2.1.3 Graph and Convolution

Convolution is originally a signal processing method, but it performs well as

a basic block for representation learning. Convolution Neural Networks (CNNs) are neural networks with multiple convolutional layers as their feature extractor. The convolutional layer performs well since it reveals the local connection of input data. Also, the computational cost is relatively low because of shared weight inside every convolutional layer.

Though the three characteristics of CNN (local connection, shared weight, ability to extend to multiple layers) are of great importance in the graph domain, CNNs are only functional on regular Euclidean data like text (1D sequence), images (2D grid), etc. Though those Euclidean data can be regarded as special instances of graph-structured data with neighbouring pixels/tokens connected, CNNs cannot operate on regular graphs without further modification since the topology structure is more complicated than Euclidean data.

2.1.4 Graph Neural Networks

The goal of GNN is to learn a hidden representation $h_v \in \mathbb{R}^s$ for every node v containing the information of itself and its neighbourhood. Unlike CNNs and RNNs stacking input features by specific orders, GNNs must traverse all the possible orders of input graphs given by edges to present them correctly. Generally, GNNs recursively update nodes' hidden state by the weighted sum of their neighbouring nodes' states, but other functions may also serve as an aggregation function. Annotating the aggregation function (transition function) as f and the function outputting the overall outcome (output function) as g, the whole node-level GNN model can be defined as:

$$h_v^t = f\left(h_v^{t-1}, h_{ne[v]}^{t-1}, w_{v,ne[v]}\right), h_v^0 = x_v$$

$$o_v = g(h_v)$$

where $ne[v]$ is the set of neighbouring nodes of v, $w_{v,ne[v]}$ is the set of

edges (with/without weighs) between v and its neighbours.

GNN can also output a graph-level result with the help of the global transition function F, and global output function G with the hidden states of all the nodes stack together as H in a compact form as:

$$H^t = F(H^{t-1})$$

$$O = G(H)$$

and parametric functions f, g, F, G can be interpreted as neural networks.

At the early stage of GNN development, parameters of transition function and output function were updated iteratively by propagating neighbouring information until a stable fixed point is reached. This method is very computationally expensive. Continuous efforts have been made to overcome this drawback, but convolutional neural networks' success encourages the proposal of a feature extractor block with shared weights for graphs. Many methods of migrating convolution to graph data are developed in parallel. They are divided into two main streams: the spectral-based approaches and the spatial-based approaches.

Bruna et al. presented the first spectral-based graph convolutional neural network in 2013 [30]. This graph convolution is based on the spectral graph theory. Since then, there has been an increasing amount of work on improving, extending, or approximating spectral-based graph convolutional networks. On the other hand, the research of spatial-based graph convolutional network started much earlier in 2009 by Micheli et al. [31]. However, the importance of this paper was overlooked, and many spatial-based graph convolutional networks emerge until recently. Apart from graph convolutional network, many other alternatives of representation learning on graphs have been developed, including graph autoencoders (GAE), recurrent graph networks

(RGN).

As the idea of convolution highly corresponds with the nature of graph-structured data, studying the robustness of graph convolutional networks may reveal more potential failures in graph neural networks. In this report, GNN will refer to graph convolutional network if not specified.

2.1.5 Taxonomy of Graph Convolutional Network

With the information of graph-structured data as inputs, the outputs of graph neural networks can be for different tasks:

- **Node-leve**l outputs relate to the node regression/classification tasks. After extracting hidden information in nodes by graph convolution, node-level tasks can be done in an end-to-end manner with help from a "head" network consisting of a multi-layer perceptron (MLP) and a SoftMax layer to produce the output.
- **Edge-level** outputs relate to the edge classification and link prediction tasks. It is achieved by comparing the similarity metrics between the hidden representation of two nodes. The similarity metric can be a neural network.
- **Graph-level** outputs relate to the graph classification tasks. Several pooling layers and readout operations are used to obtain a compact representation for the input graphs. A "head" network similar to the one in node-level GNNs is applied to generate a final prediction.

Based on the number of labelled data known to the GNN model, GNNs can also be categorized into the following groups:

- **Semi-supervised learning** (node classification): Semi-supervised learning refers to the scenario that only part of nodes in an input graph is labelled. The goal is to predict the classes of unlabelled nodes in the graph using labelled nodes as training data.

- **Supervised learning** (graph classification): This scenario is more like supervised learning in the image domain, where images can be regarded as graphs with neighbouring pixels connected. Normally, it consists of several stacked graph convolution layers for up-sampling, pooling layers for down-sampling, a readout layer to form a graph representation from node representations, and an output block of MLP and SoftMax operations.
- **Unsupervised learning** (graph embedding): Just like most embedding methods, graph embedding is also unsupervised. It can be a graph autoencoder or using a negative sampling approach.

Later in this report, supervised learning on graph classification is discussed.

2.2 History of adversarial machine learning

The very first work on adversarial machine learning dates back to 2004. In that year, Dalvi et al. showed few maliciously crafted perturbations could easily fool the linear classifiers in the content of detecting spam emails [32]. Along with the development of machine learning, adversarial machine learning (*i*) develops attacks both at training time (poisoning) and at test time (evasion); (*ii*) proposes methodologies for evaluating the security of machine learning algorithms; and (*iii*) provides robustness for machine learning models with suitable defence mechanisms.

2.2.1 Taxonomy of adversarial attacks

Adversarial attacks can invade the victim model during training time (poisoning attack) and test time (evasion attack). Based on the adversary's knowledge of the victim model, adversarial attacks can be in a white-box scenario (full knowledge), grey-box scenario (partial

knowledge) and even black-box scenario (zero-knowledge). If the objective of the adversary is to cause misclassification to specific outputs, the attack is called a targeted attack; otherwise, it is an untargeted attack.

2.2.2 Development of adversarial attacks

Though this project is on the adversarial attacks on graph-structured data, adversarial attacks in other domains like images and texts are worth mentioning since some of the methodologies apply to graphs adversarial attacks as well. Instead of a detailed survey on these algorithms, only attack algorithms related to universal attacks or graph attacks will be covered in this section as this project is on graph universal adversarial attacks.

Fast Gradient Sign Method (FGSM) [33] is the first gradient-based attack algorithm. It is a one-step algorithm and updates the adversarial perturbation by the gradient of the adversary's objective function with respect to the input in the following way:

$$x^* = x + \epsilon \cdot sign\big(\nabla_x \mathcal{L}(f_\theta(x), y)\big)$$

where f_θ is the victim model with parameters θ, x, y are the input data and label, ϵ is the pre-defined step length, and $\mathcal{L}$ is the adversary's objective function, which in this case, is defined as the negative of training objective function.

Projected Gradient Descent (PGD) [34] is an iterative variant of FGSM. It updates the perturbation gradually till the perturbation budget exceeded or the victim model successfully fooled. In every iteration, PGD updates the perturbation in the same manner as in FGSM except that the step size is set to a much smaller value.

DeepFool attack [35] is an optimization-based attack, i.e., formulating the perturbation generation process as an optimization problem instead of applying gradient directly. It comes with a closed-form representation of approximated minimum distance from the current perturbation to the closest decision boundary of the victim model as

$$\hat{l} = \arg \min_{k \neq \hat{k}(x)} \frac{|f_k'|}{||w_k'||_2}$$

where $k \in K$ is one of all possible classes, x is the benign example and

$$w_k' = \nabla f_k(x_i) - \nabla f_{\hat{k}(x)}(x_i)$$

$$f_k' = f_k(x_i) - f_{\hat{k}(x)}(x_i)$$

DeepFool updates current adversarial perturbation x_i at time step i iteratively till it successfully fools the victim model.

Carlini and Wagner's attack (C&W) [36] is another optimization-based attack. To generate subtle perturbation, the l_2 norm of perturbation acts as a penalty term in objective function instead of setting a specific threshold like in FGSM, PGD. Also, as the pixel value of images is bounded, a change of variable method is applied to ensure the generated perturbations are in the feasible domain. Overall, the objective function to optimize for C&W is:

$$\left|\left|\frac{1}{2}(\tanh w^* + 1) - x\right|\right|_2^2 + c \cdot f\left(\frac{1}{2}(\tanh w^* + 1)\right)$$

where c is a constant scaling factor found by lattice search.

2.2.3 Development of graph adversarial attacks

The taxonomy of adversarial attacks on graphs is similar to adversarial attacks on Euclidean data except that graph adversarial attacks can be further partitioned into topological attacks and node attacks based on the perturbed element of a graph.

Fast Gradient Attack (FGA) [37] is a topological white-box evasion attack on network embedding. Unlike FGSM, it is an iterative attack that toggles the connection between a pair of nodes with the highest link gradient. Link gradient is defined as the gradient of the final objective function with respect to elements of the adjacency matrix.

PGD Topology Attack [11], as its name suggests, is a topological attack inspired by PGD. It initializes a mask matrix $S \in \{0,1\}^n$ with each element of it indicating flipping an edge or not. The goal of PGD topology attack is to find

$$s^* = \arg\min \sum_{i \in \mathcal{V}} \max\left\{Z_{i,y_i} - \max_{c \neq y_i} Z_{i,c}, -k\right\}$$

where $Z_{i,c}$ is the negative cross-entropy loss between the ground truth label and class c at node i. This objective function is inspired by C&W attack. Then, S is updated in a PGD manner as follows:

$$s^t = \prod_{\mathcal{S}} [s^{t-1} - \eta \nabla f(s^{t-1})]$$

Nettack [8] is a grey-box targeted attack on graphs. It claims that the stealthy of graph perturbation should not be limited only by its norm but also by graph properties like degree distribution and feature co-occurrence. Hence, it proposes several metrics to evaluate the stealthy of

perturbation, including degree distribution preserving and node feature preserving measures as follows:

$$\Lambda(G^0, G') = -2l(D_{G^0} \cup D_{G'}) + 2\big(l(D_{G^0}) + l(D_{G'})\big)$$

$$p(i|S_u) = \frac{1}{|S_u|} \sum_{j \in S_u} \frac{1}{d_j} E_{ij}$$

where

$$\alpha_G \approx 1 + |D_G| \cdot \left[\sum_{d_i \in D_G} \log \frac{d_i}{d_{min} - \frac{1}{2}} \right]$$

$$l(D_x) = |D_x| \log \alpha_x + |D_x| \alpha_x \log d_{min} + (\alpha_x + 1) \sum_{(d_i \in D_x)} \log d_i$$

Then Nettack trains a surrogate model with linear activation function, define surrogate loss as

$$\mathcal{L}_s = \max_{c \neq c_{old}} \left[\overline{A}^2 XW \right]_{v_o c} - \left[\overline{A}^2 XW \right]_{v_o c_{old}}$$

and greedily selects perturbation from candidate sets based on the surrogate loss until budget exceeded.

Mettack [7] is a topological untargeted poisoning attack. As poisoning attacks are trying to solve a bi-level optimization problem, Mettack calculates the meta-gradient of the objective function with respect to input graphs for optimization. As the calculation of meta-gradient is computationally and memory expensive, Mettack also adapts several time-efficient and memory-efficient approximation for it.

2.2.4 Universal adversarial attacks on images and texts

Universal adversarial attack is a variant of evasion attacks. It generates a single perturbation for all the examples in a dataset, causing a high error rate. Currently, there are two main steams for universal adversarial attacks on images: data-driven attacks (white-box) and data-free attacks (grey-box).

For data-driven universal attacks, Universal Adversarial Perturbation (UAP) [16] utilizes the DeepFool algorithm to update a randomly initialized perturbation. At the same time, Network for Adversary Generation (NAG) [38] and UAP-GM (UAP with generative model) [20] train models taking random vectors as input and output universal perturbations. This kind of model is called the generative models. The objective function of the generative model in UAP-GM is in C&W-style, and it is defined as follows:

$$\mathcal{L} = \max\left\{\max_{\mathrm{i}\neq\mathrm{c}} \log[f(\delta' + x)]_i - \log[f(\delta' + x)]_c - k\right\} + \alpha \cdot \left|\left|\delta'\right|\right|_p$$

NAG introduces fooling loss $\mathcal{L}_f$ and diversity loss $\mathcal{L}_d$ to ensure the variety and effectiveness of errors caused by one single perturbation as

$$\mathcal{L}_f = -\log\left(1 - f_{c_0}^{\frac{s}{m}}\right)$$

$$\mathcal{L}_d = -\sum_{n=1}^{B} d\left(f^i(x_n + \delta_n),\ f^i(x_n + \delta_n')\right)$$

where $x_n + \delta_n'$ is the nth example in a shuffled adversarial batch.

As the victim model's training data is highly valuable and less likely to leak in a real-world scenario, data-free universal attacks achieve destructive effect while no training data is known. There are also two

steams of data-free universal attacks. Fast Feature Fool (FFF) [17] and Generative Data-free UAP (GD-UAP) [18] utilize the hidden representation of the input in the intermediate layers of the victim model. FFF randomly initializes a perturbation, feeds into the victim model, and updates the perturbation by minimizing

$$\mathcal{L} = -\log\left(\prod_{i=1}^{k} \hat{f}^i(\delta)\right)$$

where $\hat{f}^i(\delta)$ is the hidden representation of current perturbation δ as the ith hidden layer. GD-UAP trains a generator with a similar objective function except the l_2 norm of hidden representation is used to ensure the stealthy, i.e.,

$$\mathcal{L} = -log\left(\prod_{i=1}^{K} \left|\left|f^i(\delta)\right|\right|_2\right)$$

The other stream of data-free universal attack generates surrogate data. Ask, Acquire and Attack (AAA) [19] is a representative of it. It creates surrogate data by optimizing randomly initialized inputs so that they can be correctly classified. After that, a compound objective function with fooling loss and diversity loss is applied as in NAG. The specific data generation method in AAA will be discussed in the following section.

As universal attack algorithms mentioned previously are all on images with continuous feature value while some of the graphs have discrete values, universal attacks on texts, which is also discrete, may inspire our research. The representative of universal attack in NLP domain is Universal Adversarial Trigger (UAT) [15]. UAT concatenates an adversarial trigger with several tokens at the beginning or the end of input to cause a targeted wrong prediction. As the tokens are in a discrete space, optimizing them directly may not be effective at all, and

the complexity of an exhaustive search is $O(n^3)$. In UAT, a HotFlip-inspired token replacement strategy is applied. For every token adv_i in the current trigger, it selects the best replacement for it with help from token embedding e_{adv_i} as follows:

$$\arg\min_{e_i' \in \mathcal{V}} [e_i' - e_{adv_i}]^T \nabla_{e_{adv_i}} \mathcal{L}$$

where $\mathcal{V}$ is the set of all possible tokens and $\nabla_{e_{adv_i}} \mathcal{L}$ is the average gradient over a batch. By applying this gradient-guided selection algorithm on every token in the adversarial trigger, the time complexity drops to $O(n)$. Furthermore, it also uses beam search and Taylor approximation to speed up the process without lowering effectiveness.

2.2.5 Universal adversarial attack on graphs

To the best of our knowledge, there are only two universal adversarial attacks on graphs so far, and both are on node-level GNNs.

Zang et al.[23] propose a topological universal attack in Feb 2020. It flips the connections between the target node and several anchor nodes to perform an adversarial attack. The anchor nodes are computed by updating a mask vector $p \in [0,1]^n$ in a DeepFool manner where n is the total number of nodes in the graph. More specifically, in every iteration of optimization for p, a perturbed graph given current p is generated, then p is updated by DeepFool algorithm, after that, l_2 norm projection and clipping are used to ensure the updated p are in the range between 0 to 1 and also to ensure that there will not be too many anchor nodes. At the end of an iteration, a threshold of 0.5 is applied to p to make it a binary vector with 1 indicating the anchor nodes and 0 for non-anchor nodes. The misclassification rate is calculated afterwards. This iterative process will continue until the misclassification rate is high

enough or the number of iterations reaches a pre-defined maximum number.

Several months after Zang et al.'s work, Dai et al. [22] propose a targeted graph attack appending adversarial nodes on the neighbouring nodes of the target node. As the adversarial nodes are the same, this attack is also universal. It initializes several adversarial nodes with features being all zeros, update the features of adversarial nodes by the gradient of its objective function with respect to them. The objective function in this algorithm is defined as follows:

$$\mathcal{F}(A', X', v) = \left[f\left(A'_{v,v_A}, X'\right)\right]_{v,c_A} - \left[f\left(A'_{v,v_A}, X'\right)\right]_{v,c_V}$$

where v is one of the adversarial nodes, A'_{v,v_A} stands for the perturbed adjacency matrix after appending the adversarial node v_A, $[f(\cdot)]_{v,c_A}$ and $[f(\cdot)]_{v,c_v}$ refer to the output probabilities of the victim model given perturbed input on the target class c_A and current class c_V respectively.

Though these graph universal adversarial attacks (GUAAs) achieve a high attack success rate on node classification datasets like Cora, Citeseer and Pol. Blogs, the robustness of graph-level GNNs is still unknown.

2.3 Surrogate data generation methods

The goal of surrogate data generation is not to output naturally looking vivid data but to output data that the victim model can classify with high confidence. To formulate it compactly, the surrogate data generation can be represented as follows:

$$find\ x\ s.t.\ f_\theta(x) = t$$

This generation process is similar to feature visualization methods, where surrogate data are generated to understand the input patterns that a neuron corresponds to.

Intuitively, optimizing the normalized output (after a SoftMax layer) is the obvious choice. However, previous research [39] has shown that directly optimizing at the normalized output leads to an increase in the probability by reducing the pre-SoftMax logits for other classes instead of increasing one of the desired class. Hence, this optimization is often performed on the pre-SoftMax logits.

Since the objective is to generate data that can be classified with high confidence, the more straightforward approach is to optimize input data based on the output logits of the model, i.e., optimizing the following $f(\theta; x, t)$ where model parameters θ are normally frozen and require no gradient computation to save computational power, x is the current input, and t is the target outcome for x.

Based on the naïve implementation using only the objective function, there are many modifications done to improve the performance. [39] proposed to add a regularized l_2 normalization as a penalty term to optimize, i.e., the objective function becomes:

$$f(\theta; x, t) - \lambda\left|\left|x\right|\right|_2^2$$

where λ is the regularizing constant. Furthermore, [19] proposed to apply image transformations on input data to improve the robustness of surrogate data. They applied differentiable transformations like random rotation, random scaling, colour jittering, random cropping and random noises to input data independently and took the average logits for optimization. Hence, the

objective function becomes: $\sum_{T\in\mathcal{T}} f(\theta; x^T, t)$ where x^T is the input data after transformation T and $\mathcal{T}$ is the pool of all transformations.

Chapter 3

Methodology

3.1 Surrogate data generation

To migrate current surrogate data generation methods from images to graphs, there are two main problems to solve:

1. The topological structure of generated graphs

   For the topological structure of generated graphs, the first approach we implemented is to randomly initialize the adjacency matrix of adversarial nodes and then update the node features as a continuous optimization problem. However, as the victim model is trained on PROTEINS dataset and the values of node features are physically meaningful, directly optimizing them may result in producing physically unfeasible nodes. Hence, we decided to assume the adversary's knowledge of possible candidates for nodes as the information of SSEs is finite and easily accessible.

   With this assumption, one may naturally think about modifying the edges in a complete graph consists of several randomly selected nodes and take the largest connected component of it as the surrogate data. As the edge index vector X is not differentiable with respect to the objective function, modifying the edge weights may perform the same role as the node-wise updating formulation in a GCN is given by:

$$x_i' = \Theta \sum_{j \in \mathcal{N}(v) \cup \{i\}} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} x_j$$

where $e_{j,i}$ is the edge weight between node j and node i, $\widehat{d}_i = 1 + \sum_{j \in \mathcal{N}(i)} e_{j,i}$. It is obvious that having $e_{j,i} = 0$ is identical with no connection between node i and j, and vice versa. Hence, by optimizing the edge weights of a complete graph and binarizing it, the goal of optimizing the topological structure is achieved.

2. The similarity between generated data and benign data

   For previous surrogate data generation algorithms on images, distance metrics like l_p norms are used to generate images comparable to benign images. However, in this case, as the ultimate goal of surrogate graphs is to be part of the universal adversarial attacks, pursuing high similarity between surrogate data and benign data is not a critical problem. Also, since the adversarial nodes are from the same candidate pool as the benign data under the assumption made in previous paragraphs, there will not be an unseen node to the victim model in the surrogate data.

After solving these two problems, only the optimization objective function is left to deal with. As suggested in [19], surrogate data generated in different confidence level is more realistic, and the attack algorithms may perform better on this set of surrogate data. However, objective functions involving image transformations and l_2 norm of pixel values proposed in [19] does not apply to this project as only the topological structure is being optimized.

The original surrogate data generation method mentioned in [39] with variable target confidence level is selected. The objective function is the training objective function given the target class as follows:

$$\mathcal{L}_{sur} = \mathcal{L}_{train}(G, y_{tar})$$

where G is the current optimizing graph and y_{tar} is the targeted class. Also, as the objective function for training is often known to the public, and

even if it is unknown, there are certain working objective functions for certain kind of optimization problems like *Cross-Entropy Loss* or *Negative Log-Likelihood Loss* for classification problems and *Mean Squared Error Loss* for regression problems, assuming the training objective known to the adversary is not against the grey-box scenario settings. This optimization process will end when a pre-defined maximum number of iterations is reached, or the confidence of the surrogate graph becomes higher than the current threshold. An Adam optimizer is selected for optimization.

Overall, the surrogate data generation algorithm is shown in algo.1.

Algorithm 1: Surrogate Data Generation

Input: Victim model f_θ, objective function $\mathcal{L}_{sur}$, number of required surrogate data per class N, possible labels Y, learning rate η

create an empty set X

for every y_{cur} in Y:

    create an empty set $X_{y_{cur}}$

    while $i \leq N$:

        randomize an input x_i^0

        optimizer $optim = Adam(x_i, \eta)$

        feed into victim model to get output $pred_i = f_\theta(x_i)$

        t = 0

        $tar = random(0.65,0.99)$

        while $pred_i[y_{cur}] \leq tar$ or $t = max\_iter$:

            calculate $\mathcal{L}_{sur} = \mathcal{L}_{train}(x_i^t, y_i)$

            update $x_i^{t+1} = optim\left(\nabla_{x_i^t}\mathcal{L}_{sur}, x_i^t\right)$

            $t = t + 1$

        end while

        append x_i^t to $X_{y_{cur}}$

    end while

    append $X_{y_{cur}}$ to X

end for

Output: a set of surrogate data X

3.2 Universal Adversarial Attack Algorithm

We observe that given the candidate set of node features, updating the feature of adversarial nodes will be similar to finding adversarial tokens in NLP except that the topological connections between adversarial nodes and the position to append adversarial nodes are still undecided.

For topological structure, as the number of adversarial nodes is very limited (maximum at three) and the node features in adversarial nodes are changing during optimization, setting an adjacency matrix of adversarial node randomly at the beginning of adversarial attack does not affect the attack outcome. Hence, in this project, a randomly initialized adjacency matrix is generated first, and the following optimization is on node features only.

A heuristic thought on position to append the generated nodes is to append it to the node with the greatest influence. To identify the importance of a node mathematically, a concept called "vertex centrality" from graph theory is adopted. There are many implementations for centrality, including degree centrality, betweenness centrality, eigenvector centrality and so on as explained in Chapter.2. Several appending strategies are evaluated throughout the experiment.

The algorithm in this project is adapted from [15] and it is shown in algo.3. However, in the original algorithm in [15] for universal adversarial perturbation in NLP domain, the embedding of tokens is independent of its neighbouring tokens. It can be extracted easily by sparse matrix multiplication between index tensor and embedding weights. However, the embedding of nodes is dependent on its neighbours, and the node feature is not as representative as the embedding. To tackle this problem, node embedding can be approximated by the difference between the embedding of two graphs in which one graph is appended with the evaluating node while

the other is the same except it is without the evaluating node. However, experiments have shown that this approach is very computationally intensive. Furthermore, the number of adversarial nodes is limited to 1 under this circumstance as the same adversarial node may have different embedding if the topological structure of adversarial nodes is different. The detailed implementation of this node embedding via the difference of graph embeddings (DGE) method is shown in algo.2. A straightforward heuristic replacement of that is to use the unprocessed node features (NF method) as the node embedding. NF method brings a significant decrease in computational cost compared to the DGE method, and the number of adversarial nodes is now not restricted. Both approaches are tested in our experiments.

Algorithm 2: Node Embedding via Difference of Graph Embeddings
Input: adversarial node V_{adv} with feature X_{adv}, victim model f_θ, the current batch of training data for the adversary G_{train}^{B}
$G_{adv} = GraphAppend(G_{train}^{B}, V_{adv})$ get graph embedding E_{adv} during forward propagation $f_\theta(G_{adv})$ get graph embedding E_{ori} during forward propagation $f_\theta(G_{train}^{B})$ node embedding $E_{V_{adv}} = E_{adv} - E_{ori}$
Output: Embedding of adversarial node $E_{V_{adv}}$

To evaluate the effect of adversarial attacks, the classification accuracy of the victim model is calculated. It is the ratio of the number of examples correctly predicted by the victim model and the total number of examples. As the victim model in this project is a binary classifier, attacks that decrease the classification accuracy to around 50% can be considered a successful attack.

Algorithm 3: Hot-flip Inspired Universal Adversarial Attack on Graphs

Input: Number of adversarial nodes n, victim model f_θ, set of all possible nodes $\mathcal{V}$, training data G_{train} for the adversary (including their labels y), the objective function of the adversary $\mathcal{L}_{adv}$

randomly select n adversarial nodes V_{adv} from $\mathcal{V}$

for every batch G_{train}^{B} of G_{train}:

  $G_{adv} = GraphAppend(G_{train}^{B}, V_{adv})$

  perform a forward propagation $pred = f_\theta(G_{adv})$

  compute the value of objective function $\mathcal{L} = \mathcal{L}_{adv}(f_\theta; pred, y)$

  for every node V_{adv_i} in V_{adv}:

    get node embedding $E_{V_{adv_i}}$

    for every node V_i in $\mathcal{V}$:

      get node embedding E_{v_i}

      compute the score $\mathcal{S} = \left[E_{v_i} - E_{V_{adv_i}}\right]^T \nabla_{E_{V_{adv_i}}} \mathcal{L}$

      record $\mathcal{S}$ for V_i

    end for

  end for

  replace V_{adv_i} with V_i with the lowest $\mathcal{S}$

end for

Output: Adversarial Nodes V_{adv}

3.3 Graph Appending Process

As the adversarial nodes are to be appended into a node in the benign graph, it is necessary to have a function that appends nodes to a graph. Though there is a set of functions in *NetworkX* [40] providing node or edge modification operations, they are not batched. Creating such a function by ourselves may also leave more room for customization, which is essential in adversarial attacks.

The appending process is basically slicing the original matrices into two halves, concatenating these two halves and the matrices from adversarial nodes together. This procedure is not differentiable, not due to the fact that it involves operations like inserting, slicing and concatenating, it is because a copy of the original graph is created. The following operations are all performed on this copy to avoid unseen modifications on the original data due to some potential in-place operations.

Also, as this project is built on *torch-geometric*[41], a library for geometric deep learning and the edges in *torch-geometric* are not represented in adjacency matrices, but in a matrix in shape of *(2, number of edges)* which each row contains the node indexes of the two ends of an edge. Hence, functions transforming an adjacency matrix into this "edge index" form and the its reverse function are implemented.

3.4 Overall procedure

The whole data-free graph universal adversarial attack can be divided into two phases: the surrogate data generation phase and the adversarial nodes generation phase.

In the surrogate data generation phase, surrogate data are generated according to the possible labels the victim model might predict. It starts from randomly initialized data, updated with reference from the target class and the predicted logits given by the victim model.

After the surrogate data generation phase, the attack enters the adversarial nodes generation phase. The proposed universal adversarial attack algorithm is performed on the surrogate data generated before, and a graph consists of one or more adversarial nodes is produced.

The graph of adversarial nodes will be appended to every benign data in the test dataset to perform the attack. This fainted set of data will be fed into the victim model and compare the predicted outcome and their corresponding labels. A lower classification accuracy is expected for a successful adversarial attack. The overall pipeline for this data-free universal adversarial attack on graph-level GNNs is shown in Fig.1.

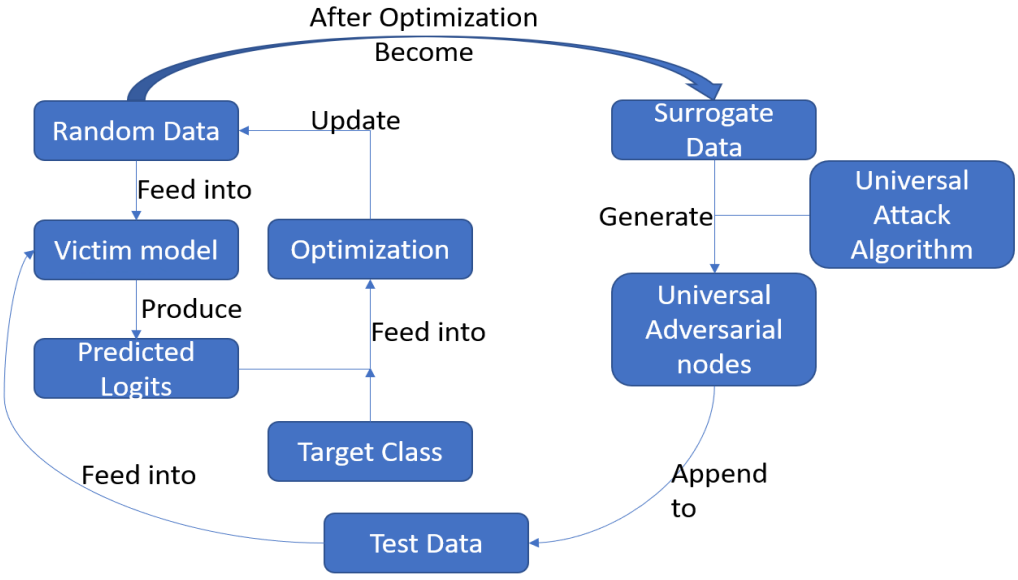

Figure 1 Overall Universal Attack Pipeline

Chapter 4

Experiments

4.1 Experiment setup

4.1.1 Dataset

PROTEINS dataset is selected to evaluate the robustness of GNN. It is from TUDatasets, a collection of benchmark datasets for classification and regression, and it is commonly used in the evaluation of the representation learning capability of GNNs. It contains 1113 unweighted graphs, each graph representing a protein molecule with nodes representing secondary structure elements (SSEs) and two nodes are connected by an edge if they are neighbours in either the amino-acid sequence or 3D space. The binary label in PROTEINS dataset suggests whether a protein is a non-enzyme. The dataset is further split into training, validation and testing set at the ratio of 8:1:1.

Surrogate data are generated with assuming SSEs known to the adversary. 500 surrogate data entries are generated for every class with variable confidence level from 0.65 to 0.99. Hence in total, there are 1,000 surrogate data entries.

4.1.2 Model

The victim model for this proposed data-free universal adversarial attack is GNN with Hierarchical Graph Pooling and Structure Learning (HGP-SL)[42], the current state-of-the-art method of graph classification on PROTEINS dataset. The brief structure of it is shown in Fig. 2. As Zhang et

al., the authors of HGP-SL, open-sourced their implementation for HGP-SL [42], we adopt their implementation as well as their hyper-parameter settings. An Adam optimizer is applied with a learning rate set to 0.001 and weight decay set to 0.001. At each iteration of optimization, a mini-batch of 512 data entries is fed into the model. Also, early stopping technique is applied to prevent the victim model from overfitting. The early-stopping patience is set to 100, i.e., if the model fails to perform better (judged by the value of training objective function on validation data) within 100 epochs, the training process will end and the model with the best performance will be saved. The detailed hyperparameters are shown in Table.1.

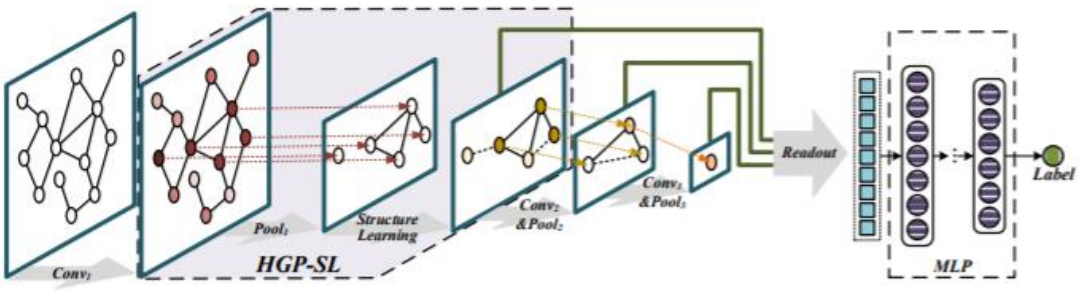

Figure 2 Model Structure of HGP-SL, adopted from [42]

Table 1 Hyper-parameters for Training HGP-SL as Victim Model

Learning rate	Weight decay	Batch size	Pooling ratio	Dropout ratio	Hidden Size	No. of layers	ES Patience
0.001	0.001	512	0.5	0	128	3	100

To assess the efficiency of the proposed algorithm, all the computations, including surrogate data generation, training the victim model, universal adversarial attacks, are performed on a server with Intel Xeon W-2133 CPU and 64 Gigabyte of RAM. An RTX-2080ti graphics card is used to accelerate computations with CUDA implementation of them.

Table 2 Classification Accuracy under Random Selection Attack

Attack Scenario			Accuracy
Random Selection (Baseline)	$\|\mathcal{V}_{adv}\|$=1	Degree Centrality	74.11%
		Degree Centrality (minimum)	76.79%
		Betweenness Centrality	73.57%
		Eigenvector Centrality	76.96%
	$\|\mathcal{V}_{adv}\|$=2	Degree Centrality	76.25%
		Degree Centrality (minimum)	75.36%
		Betweenness Centrality	73.04%
		Eigenvector Centrality	71.96%
	$\|\mathcal{V}_{adv}\|$=3	Degree Centrality	72.50%
		Degree Centrality (minimum)	74.82%
		Betweenness Centrality	74.29%
		Eigenvector Centrality	74.29%

Table 3 Classification Accuracy under Different Attack Scenarios

Attack Scenario			Accuracy
Benign			80.13%
Graph Embedding		Degree Centrality	62.50%
		Degree Centrality (minimum)	66.96%
		Betweenness Centrality	63.39%
		Eigenvector Centrality	69.64%
Node feature + benign data	$\|\mathcal{V}_{adv}\|$=1	Degree Centrality	62.50%
		Degree Centrality (minimum)	66.96%
		Betweenness Centrality	61.61%
		Eigenvector Centrality	65.18%
	$\|\mathcal{V}_{adv}\|$=2	Degree Centrality	62.50%
		Degree Centrality (minimum)	65.18%
		Betweenness Centrality	57.14%
		Eigenvector Centrality	62.50%
	$\|\mathcal{V}_{adv}\|$=3	Degree Centrality	61.61%
		Degree Centrality (minimum)	71.43%
		Betweenness Centrality	**52.68%**
		Eigenvector Centrality	68.75%
Node feature + class impression	$\|\mathcal{V}_{adv}\|$=1	Degree Centrality	63.39%
		Degree Centrality (minimum)	**58.93%**
		Betweenness Centrality	59.82%
		Eigenvector Centrality	60.71%
	$\|\mathcal{V}_{adv}\|$=2	Degree Centrality	**55.36%**
		Degree Centrality (minimum)	65.18%
		Betweenness Centrality	58.04%
		Eigenvector Centrality	58.04%
	$\|\mathcal{V}_{adv}\|$=3	Degree Centrality	77.68%
		Degree Centrality (minimum)	66.07%
		Betweenness Centrality	58.93%
		Eigenvector Centrality	64.29%

4.2 Result and Discussion

All the classification accuracy of the victim model under different experiment settings are listed in Table.3. In the following sections, $|\mathcal{V}_{adv}|$ will refer to the number of adversarial nodes if not specified. Under every $|\mathcal{V}_{adv}|$ setting, the worst classification accuracy is bolded. Adversarial nodes selected randomly are chosen as the baseline method as there is no graph-level universal adversarial attack yet. This baseline method is annotated as "Random Selection Attack", it is repeated five times to be unbiased, and the result of Random Selection Attack is shown in Table.2.

4.2.1 Node Embedding Methods Comparison

As mentioned in Chapter.3.2, the embedding of nodes can be retrieved from the difference between two graphs or simply the node features. In the following sections, node embedding via difference of graph embedding will be annotated as DGE method while the node feature as node embedding will be annotated as NF method. Several experiments under different centrality measures and node embedding methods are performed to study the effectiveness of DGE and NF node embedding methods. The number of

adversarial nodes by the NF method is set to 1 to control the variables in this experiment. The result of this experiment is shown in Fig.3.

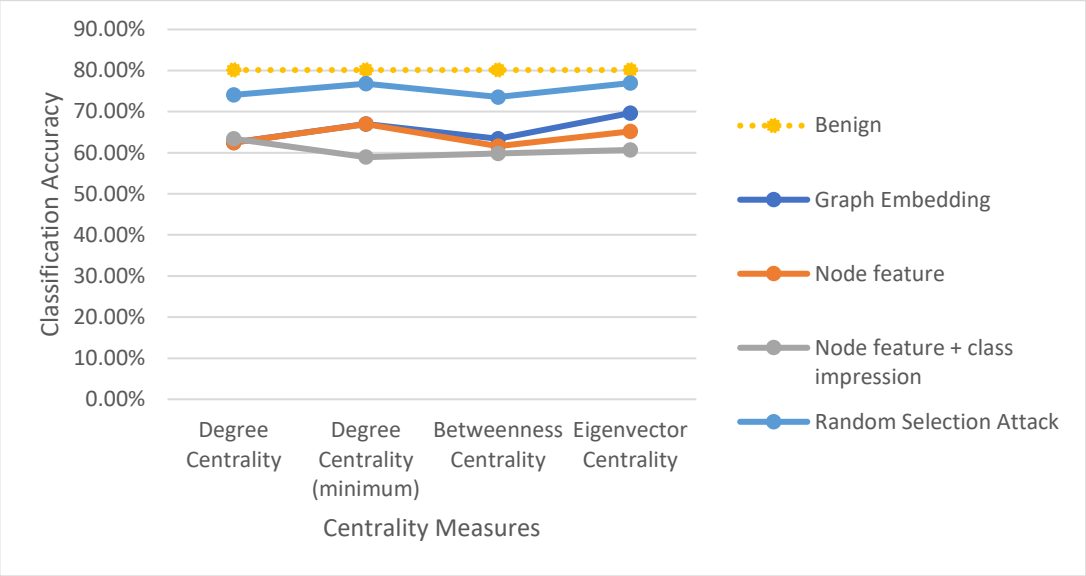

Figure 3 Comparison of node embedding methods.

It is obvious that the classification accuracy of the victim model given adversarial nodes generated with the DGE method as node embedding is higher than those generated with NF as node embedding. This suggests that the DGE method is not as representative as the NF method in an adversarial attack context.

Comparing the two node embedding methods, though the DGE method is more heuristically reasonable as it takes the whole graph into consideration and the node feature is just a 4-dimensional vector containing an integer and a one-hot vector representing the van de wall force. However, the NF method as node embedding outperforms the DGE method both in performance and efficiency. For the performance part, it may be since the gradient of the objective function with respect to embedding is also calculated by the difference between two gradients with respect to corresponding graphs in the DGE method. For the efficiency part, on average, it takes 29.04 seconds to generate the universal adversarial node using the NF method, while it takes

899.73 seconds for the DGE method on the server mentioned above. The efficiency gap is due to the fact that it takes two complete graph embedding forward and backward propagations and an appending operation for the DGE method, while it only takes a simple slicing operation to get the node feature in the NF method.

4.2.2 Data-free Universal Adversarial Attack

To evaluate the similarity between generated surrogate data and benign data in terms of the effectiveness of adversarial perturbations generated, an experiment comparing adversarial perturbations generated on benign data and surrogate data is performed. Several numbers of adversarial nodes are tested to compare their performance. The classification accuracy of the victim model on the test dataset appended by two/three adversarial nodes under different centrality measures is shown in Fig. 4 and Fig.5.

Comparing the results, the average classification accuracy of the victim model given two adversarial nodes is 61.83% on actual training data and 59.15% on surrogate data, while given three adversarial nodes, the average classification accuracy becomes 63.62% on actual training data and 66.74% on surrogate data.

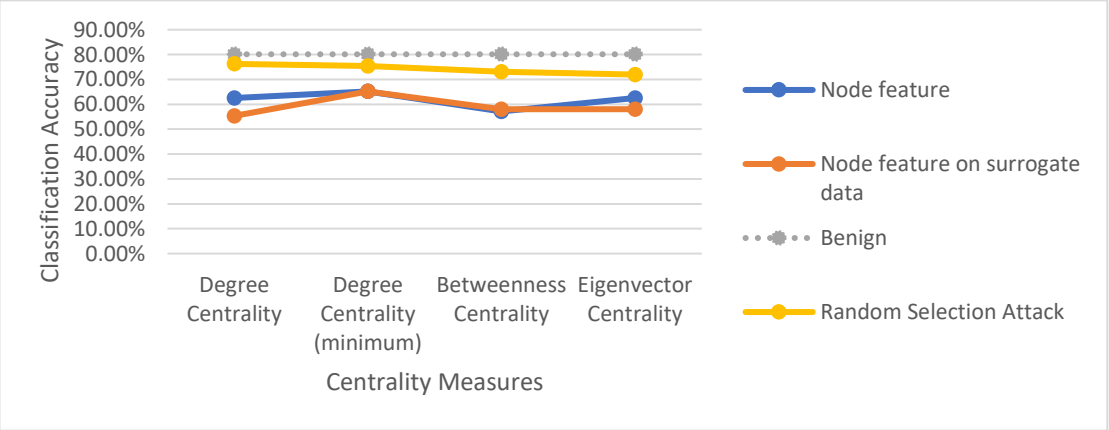

Figure 4 Classification Accuracy Given Two Adversarial Nodes

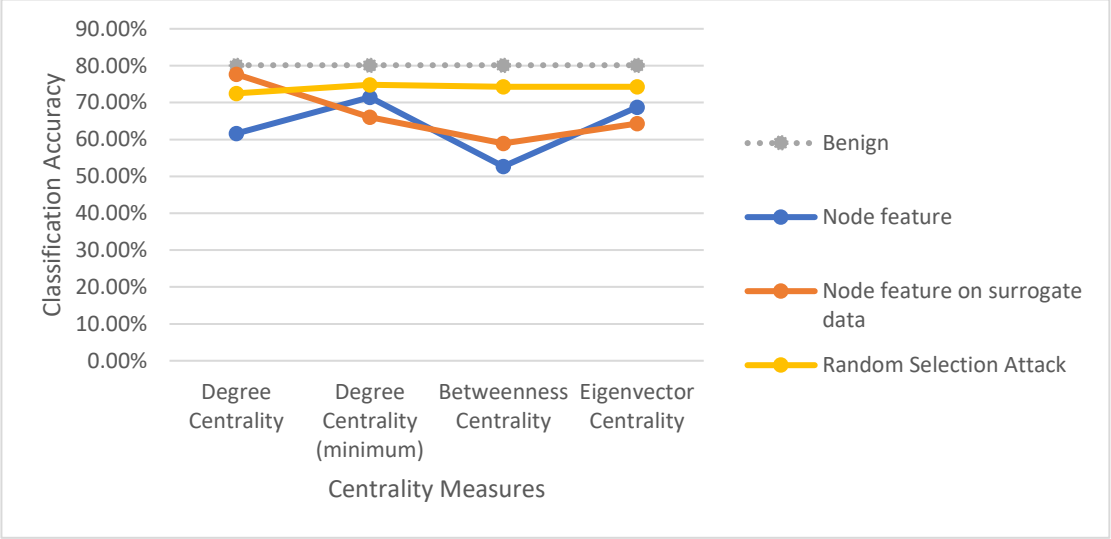

Figure 5 Classification Accuracy Given Three Adversarial Nodes

It is reasonable that the adversarial attack trained on actual training data decreases the classification accuracy to a lower value than the one trained on surrogate data since the surrogate data is just a mimic of them. However, it is still fascinating to know that a data-free grey-box attack outperforms a white-box attack under certain circumstances. This can be explained by that surrogate data reveals the hidden common features in the dataset while benign data might be too specific and overlook several features.

Though the heuristic thought on accuracy given two or three adversarial nodes is that more adversarial nodes will boost the attack performance, our experiments do not prove this idea. Considering that this dataset is to predict whether a protein is an enzyme into consideration, it is reasonable to have accuracies given three adversarial nodes become higher than those given two adversarial nodes under certain circumstances since appending more nodes breaks the original degree distribution and may make the classification even easier.

4.2.3 Comparison between centrality measures

As suggested in [43], centrality measures are highly related to the performance of adversarial attacks. Hence, different centrality measures, including Degree Centrality (deg), Betweenness Centrality (btw), and Eigenvector Centrality (eig), are applied to experiments. Also, to provide another point of view, appending adversarial nodes to the node with the lowest degree centrality is applied.

The average classification accuracies of the victim model over node embedding methods under different centrality measures are shown in Table.4 with the lowest accuracy (the best attack performance) in every centrality measure settings being bolded. It is obvious that the attack using minimum degree centrality yields the worst attack performance by having the highest classification accuracy under two $|\mathcal{V}_{adv}|$ settings out of three, which further proves the findings in [43]. Also, on average, attacks using the Betweenness Centrality measure to append adversarial nodes result in the lowest classification accuracies under all the settings, suggesting that betweenness centrality may reflect the importance of the node better than degree Eigenvector centrality in this context.

4.2.4 Comparison with the baseline method

Comparing the best result among all centrality measures with baseline method Random Selection Attack, it is obvious that the proposed attack in the project decreases the classification accuracy of the victim model to a much lower level, and this performance gap widens as the $|\mathcal{V}_{adv}|$ increases. Given $|\mathcal{V}_{adv}| = 1$, Random Selection Attack lowers down the accuracy of 75.36% while proposed GUAA yields an accuracy of 61.61%; given $|\mathcal{V}_{adv}| = 2$, Random Selection Attack can only lower down the accuracy of 74.15% while proposed GUAA is able to decrease the accuracy to 61.61%;

when $|\mathcal{V}_{adv}| = 3$, this gap becomes 73.97% for Random Selection Attack and 55.80% for proposed GUAA.

In Fig.3-5, Random Selection Attack also fails to outperform our proposed GUAA under various scenarios except for ($|\mathcal{V}_{adv}| = 3$, degree centrality measure) setting.

The gap between Random Selection Attack and the proposed GUAA indicates that appending randomly selected nodes does not have a significant adversarial effect, suggesting that the decrease of classification accuracy does not come from the change of degree distribution or other graph properties, which further proves the effectiveness of the GUAA proposed in this project.

Table 4 Average Classification Accuracy under Different Centrality Measures

$\lvert\mathcal{V}_{adv}\rvert$ / Centrality Measure	$\lvert\mathcal{V}_{adv}\rvert = 1$	$\lvert\mathcal{V}_{adv}\rvert = 2$	$\lvert\mathcal{V}_{adv}\rvert = 3$
Benign	80.13%		
Random Selection Attack	75.36%	74.15%	73.97%
Degree Centrality	62.80%	58.93%	69.64%
Degree Centrality(min)	64.29%	65.18%	68.75%
Betweenness Centrality	**61.61%**	**57.59%**	**55.80%**
Eigenvector Centrality	65.18%	60.27%	66.52%

Chapter 5 Conclusions and Future Work

In this project, a data-free universal adversarial attack algorithm on graph-level GNNs is proposed. This project fills the current research gap by evaluating the performance of graph-level GNNs under universal adversarial attacks in both white-box and grey-box scenario. The proposed algorithm brings significant and destructive effect to graph-level GNNs, to be specific, lowers down the classification accuracy of the current state-of-the-art model to a random-guessing level, provides a stronger opponent for further robustness studies.

This work can be extended to node-level GNNs by adapting new algorithms to decide the position of adversarial nodes appended to. Also, surrogate data generation is applicable for node-level GNNs, which provides another powerful algorithm for studying robustness under grey-box scenarios. Moreover, algorithms proposed in this project can be used as a strong opponent against newly proposed defence mechanisms in future defence studies.

Reflection on Learning Outcome Attainment

Throughout this FYP, I have learned a lot about GNNs and adversarial machine learning. During this project, I was stuck at first trying to implement someone's algorithm, then Prof. Tay pointed out that the algorithm is not essential, and I moved on from that. This experience practised my problem analysis skill, I can focus on important tasks without being distracted in the future. As this is a research-based project, I have to discuss with Dr. Song Yang a lot about current findings and future works, which practices my communication skills.

References

[1] W. Fan *et al.*, "Graph neural networks for social recommendation," in *The World Wide Web Conference*, 2019, pp. 417-426.

[2] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *arXiv preprint arXiv:1802.09691*, 2018.

[3] D. Duvenaud *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," *arXiv preprint arXiv:1509.09292*, 2015.

[4] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning*, 2017: PMLR, pp. 1263-1272.

[5] X. Geng *et al.*, "Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, 2019, vol. 33, no. 01, pp. 3656-3663.

[6] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, vol. 33, no. 01, pp. 922-929.

[7] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," *arXiv preprint arXiv:1902.08412*, 2019.

[8] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2847-2856.

[9] Y. Sun, S. Wang, X. Tang, T.-Y. Hsieh, and V. Honavar, "Node injection attacks on graphs via reinforcement learning," *arXiv preprint arXiv:1909.06543*, 2019.

[10] X. Liu, S. Si, X. Zhu, Y. Li, and C.-J. Hsieh, "A unified framework for data poisoning attack to graph-based semi-supervised learning," *arXiv preprint arXiv:1910.14147*, 2019.

[11] K. Xu *et al.*, "Topology attack and defense for graph neural networks: An optimization perspective," *arXiv preprint arXiv:1906.04214*, 2019.

[12] B. Wang and N. Z. Gong, "Attacking graph-based classification via manipulating the graph structure," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2023-2040.

[13] X. Zhang and M. Zitnik, "Gnnguard: Defending graph neural networks against adversarial attacks," *arXiv preprint arXiv:2006.08149*, 2020.

[14] A. Bojchevski and S. Günnemann, "Certifiable robustness to graph perturbations," *arXiv preprint arXiv:1910.14356*, 2019.

[15] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh, "Universal adversarial triggers for attacking and analyzing NLP," *arXiv preprint arXiv:1908.07125*, 2019.

[16] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765-1773.

[17] K. R. Mopuri, U. Garg, and R. V. Babu, "Fast feature fool: A data independent approach to universal adversarial perturbations," *arXiv preprint arXiv:1707.05572*, 2017.

[18] K. R. Mopuri, A. Ganeshan, and R. V. Babu, "Generalizable data-free objective for crafting universal adversarial perturbations," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 10, pp. 2452-2465, 2018.

[19] K. R. Mopuri, P. K. Uppala, and R. V. Babu, "Ask, acquire, and attack: Data-free uap generation using class impressions," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19-34.

[20] J. Hayes and G. Danezis, "Learning universal adversarial perturbations with generative models," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018: IEEE, pp. 43-49.

[21] Z. Chen, L. Xie, S. Pang, Y. He, and Q. Tian, "Appending adversarial frames for universal video attack," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 3199-3208.

[22] J. Dai, W. Zhu, and X. Luo, "A Targeted Universal Attack on Graph Convolutional Network," *arXiv preprint arXiv:2011.14365*, 2020.

[23] X. Zang, Y. Xie, J. Chen, and B. Yuan, "Graph universal adversarial attacks: A few bad actors ruin graph learning models," *arXiv preprint arXiv:2002.04784*, 2020.

[24] L. Solá, M. Romance, R. Criado, J. Flores, A. García del Amo, and S. Boccaletti, "Eigenvector centrality of nodes in multiplex networks," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 23, no. 3, p. 033131, 2013.

[25] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701-710.

[26] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855-864.

[27] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067-1077.

[28] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," in *IJCAI*, 2015, vol. 2015, pp. 2111-2117.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[30] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[31] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks,* vol. 20, no. 3, pp. 498-511, 2009.

[32] N. Dalvi, P. Domingos, S. Sanghai, and D. Verma, "Adversarial classification," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 99-108.

[33] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572,* 2014.

[34] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083,* 2017.

[35] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574-2582.

[36] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, 2017: IEEE, pp. 39-57.

[37] J. Chen, Y. Wu, X. Xu, Y. Chen, H. Zheng, and Q. Xuan, "Fast gradient attack on network embedding," *arXiv preprint arXiv:1809.02797,* 2018.

[38] K. R. Mopuri, U. Ojha, U. Garg, and R. V. Babu, "NAG: Network for adversary generation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 742-751.

[39] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034,* 2013.

[40] D. A. Schult and P. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in science conferences (SciPy 2008)*, 2008, vol. 2008: Pasadena, CA, pp. 11-16.

[41] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," *arXiv preprint arXiv:1903.02428,* 2019.

[42] Z. Zhang *et al.*, "Hierarchical graph pooling with structure learning," *arXiv preprint arXiv:1911.05954,* 2019.

[43] J. Xu *et al.*, "Query-free Black-box Adversarial Attacks on Graphs," *arXiv preprint arXiv:2012.06757,* 2020.

Appendix

Table 5 Overall Experiment Results

Attack Scenario			Trigger Index	Accu	Time Cost
Benign				80.13%	
Graph Embedding		Degree Centrality	25	62.50%	638.07
		Degree Centrality (minimum)	82	66.96%	638.36
		Betweenness Centrality	82	63.39%	1462.62
		Eigenvector Centrality	57	69.64%	859.85
Node feature	$\lvert\mathcal{V}_{adv}\rvert = 1$	Degree Centrality	[80]	62.50%	26.20
		Degree Centrality (minimum)	[106]	66.96%	26.10
		Betweenness Centrality	[82]	61.61%	35.92
		Eigenvector Centrality	[82]	65.18%	27.93
	$\lvert\mathcal{V}_{adv}\rvert = 2$	Degree Centrality	[82, 82]	62.50%	31.27
		Degree Centrality (minimum)	[82, 82]	65.18%	32.06
		Betweenness Centrality	[82, 80]	57.14%	40.78
		Eigenvector Centrality	[82, 82]	62.50%	34.33
	$\lvert\mathcal{V}_{adv}\rvert = 3$	Degree Centrality	[82, 82, 82]	61.61%	35.06
		Degree Centrality (minimum)	[82, 82, 82]	71.43%	36.34
		Betweenness Centrality	[80, 80, 80]	52.68%	44.15

		Eigenvector Centrality	[80, 80, 80]	68.75%	38.28
Node feature + class impression	$\|\mathcal{V}_{adv}\| = 1$	Degree Centrality	[106]	63.39%	992.63
		Degree Centrality (minimum)	[82]	58.93%	946.71
		Betweenness Centrality	[82]	59.82%	938.46
		Eigenvector Centrality	[106]	60.71%	938.42
	$\|\mathcal{V}_{adv}\| = 2$	Degree Centrality	[106, 106]	55.36%	936.92
		Degree Centrality (minimum)	[106, 106]	65.18%	936.57
		Betweenness Centrality	[106, 106]	58.04%	937.35
		Eigenvector Centrality	[106, 106]	58.04%	934.81
	$\|\mathcal{V}_{adv}\| = 3$	Degree Centrality	[82, 82, 82]	77.68%	955.48
		Degree Centrality (minimum)	[82, 82, 82]	66.07%	954.67
		Betweenness Centrality	[80, 80, 80]	58.93%	954.62
		Eigenvector Centrality	[80, 80, 80]	64.29%	957.42