WASHINGTON UNIVERSITY IN ST. LOUIS

McKelvey School of Engineering
Department of Computer Science & Engineering

Thesis Examination Committee:
Yevgeniy Vorobeychik, Chair
Nathan Jacobs
Ning Zhang

Adversarial Attacks on Graph Embeddings Based on Text Dataset

by
Ying (Fiona) Xu

A thesis presented to
the McKelvey School of Engineering
of Washington University in
partial fulfillment of the
requirements for the degree
of Master of Science

Dec 2023
St. Louis, Missouri

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I wish to convey my profound gratitude to my advisor, Prof. Yevgeniy Vorobeychik. His invaluable advice and continuous feedback have been pivotal in bringing this project to fruition. His deep professional knowledge and rigorous approach have not only influenced my research style but also left an enduring impression on me. Beyond the confines of academia, his guidance has been a beacon as I navigated life in the USA. I am truly fortunate to anticipate continuing my research journey under his esteemed guidance in the years to come.

I extend my sincere appreciation to my committee members, Prof. Nathan Jacobs, and Prof. Ning Zhang. Their expertise, constructive criticism, and unwavering support have been integral to the success of this work.

My gratitude also goes out to my lab members, especially Michael Lanier, Anindya Sarkar, Junlin Wu, Andrew Estornell, Jinghan Yang, and Rajagopal Venkatasaramani. Their contributions have been multifaceted, aiding me not only in ideation but also in practical coding advice. Their collective experiences have been an invaluable resource to me.

A special thanks is reserved for Prof. Donsub Rim, my numerical analysis mentor. His instructions in the MATH449 course have not only deepened my understanding of numerical analysis but also inspired me with new ideas, significantly influencing my research approach.

Last but certainly not least, a heartfelt thanks to my parents and my friends at WashU. Their encouragement played a significant role in my decision to pursue the Ph.D. program and their continued support has been a source of strength throughout.

Ying (Fiona) Xu

Washington University in St. Louis
Dec 2023

ABSTRACT OF THE THESIS

Adversarial Attacks on Graph Embeddings Based on Text Dataset

by

Ying (Fiona) Xu

Master of Science in Computer Science

Washington University in St. Louis, 2023

Professor Yevgeniy Vorobeychik, Chair

This study delves into the intriguing domain of graph data vulnerability, specifically exploring the transformation from textual datasets to graph representations. Central to our investigation is the examination of the susceptibility of graph data to adversarial attacks. Contrasting with state-of-the-art approaches like Netattack and Meta Attack, which target both node attributes and graph structure, our experimental focus is primarily on attacking node embeddings.

Through empirical experimentation, we assess the effectiveness of both decision-time and poisoning attacks on graph neural networks. A significant finding of our study is that decision-time attacks employing Projected Gradient Descent (PGD) demonstrate greater efficacy compared to poisoning attacks that utilize Mean Node Embeddings and Graph Contrastive Learning. This insight contributes to the broader understanding of graph data security, highlighting critical areas where graph-based models are most vulnerable and paving the way for the development of more robust defenses.

Our research is supported by a comprehensive code repository, which includes all scripts and data used in our experiments. This repository is openly accessible and can be found at https://github.com/YingXu001/Attack_Graph, providing a valuable resource for further exploration and development in the field of graph data security.

Chapter 1

Introduction

In the contemporary era characterized by hyperconnectivity, platforms such as Reddit, Twitter, and YouTube have transcended mere utilities and are now firmly cemented as essential facets of our daily communications. These platforms, through their intricate algorithms, foster and produce a vast web of conversations and affiliations, culminating in multifaceted social network graphs. Within these graphs, every comment is not just a mere statement but a node; a point of interconnection. Especially concerning are negative interactions which, like ripples in a pond, can magnify and resonate far beyond their point of origin, emphasizing the crucial nature of decoding these networks.

Such negative sentiments often originate from personal adversities, including but not limited to emotional disorders like depression, or exposure to malicious content online [16]. These distressing comments, laden with derogatory language, become intertwined and form intricate patterns in the vast tapestry of the social network. Our mission is not merely to observe but to penetrate these multifarious links, thereby elucidating the dynamics and interrelations of noxious engagements within these platforms.

In the computational realm, the field of Natural Language Processing (NLP) has undergone a metamorphosis, evolving rapidly and spearheading innovations in tasks related to text categorization. Further augmenting our knowledge reservoir is the advent of Graph Neural Networks (GNNs), which have immensely enhanced our comprehension of structured information. By merging the prowess of NLP paradigms with the capabilities of GNNs, we stand on the brink of a novel convergence. This confluence augments text classification techniques, a phenomenon articulated by Huang et al. [8].

Moreover, the exploration of adversarial attacks on graphs, which involves altering the graph's structure and node attribute as introduced by Zügner et al. [25], is gaining traction. We introduce this methodology to the context of social networks, focusing on the

modification of node embeddings to mitigate the impact of such attacks. Our aim is to weaken the efficacy of adversarial strategies while still representing the significant effects they have during both the poisoning and decision phases of the attack.

Chapter 2

Related Work & Prelimilaries

This chapter delves into key concepts and prior research that lay the groundwork for the present study, elucidating the principles of Machine Learning (ML), its adversarial counterpart, Graph Neural Networks (GNN), and the linguistic marvel that is Natural Language Processing (NLP). These elements are pivotal in shaping the contours of my research project.

2.1 Machine Learning

Machine Learning (ML) has evolved to be a key component of modern computational research, weaving together mathematical algorithms and vast amounts of data to bestow machines with the ability to learn patterns and make decisions. Its applications span diverse domains such as social networking[1], recommendation systems[11], and natural language processing[20]. ML can be broadly classified into categories based on the nature and extent of supervision required during the model's training phase. The subsequent sections will examine the three primary ML paradigms—supervised, semi-supervised, and unsupervised learning—highlighting significant contributions and methodologies within each.

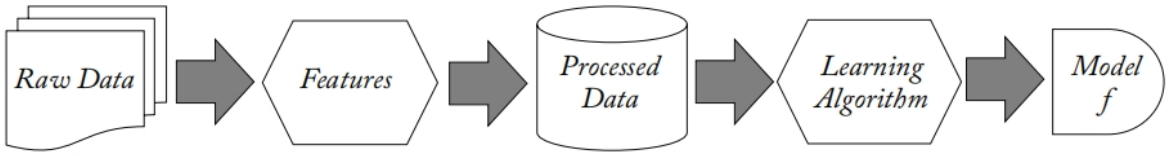

Figure 2.1: A schematic view of the machine learning workflow illustrating the transition from raw data to a functional model. Cite from the book "Adversarial Machine Learning" [16].

Before delving into these classifications, it is essential to understand the typical workflow of machine learning processes 2.1. Initially, raw datasets are amassed, from which relevant

features—denoted as x_i—are extracted through various feature extraction techniques. The data must then undergo preprocessing, such as normalization, to ensure compatibility with the learning algorithms. Upon selecting an appropriate learning algorithm, a predictive model f is constructed.

The ensuing sections will delineate the three categories of machine learning: supervised learning, where the model is trained with labeled data; semi-supervised learning, which utilizes a mix of labeled and unlabeled data; and unsupervised learning, which draws patterns from data without preassigned labels.

2.1.1 Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples [3]. In the dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$, $\mathbf{x}_i$ denotes the input features and y_i denotes the corresponding labels [16]. The aim is to learn a mapping $f : \mathbf{x}_i \to y_i$ that can predict the label y of a new, unseen instance $\mathbf{x}$.

A commonly used loss function in supervised learning is the mean squared error (MSE) for regression problems, given by:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2 \tag{2.1}$$

For classification problems, the cross-entropy loss for multi-class classification is commonly used, defined as:

$$\text{Cross-Entropy Loss} = -\sum_{i=1}^{n}\sum_{c=1}^{K} y_{i,c}\log(f(\mathbf{x}_i)_c) \tag{2.2}$$

where n is the number of training examples, K is the number of classes, $y_{i,c}$ is a binary indicator of whether class label c is the correct classification for observation i, and $f(\mathbf{x}_i)_c$ is the predicted probability that observation i belongs to class c.

2.1.2 Semi-supervised Learning

Semi-supervised learning involves a combination of a small amount of labeled data and a large amount of unlabeled data during training. The dataset for semi-supervised learning can be represented as $\mathcal{D}_L = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ and $\mathcal{D}_U = \{\mathbf{x}_j\}_{j=l+1}^{u}$, where $\mathcal{D}_L$ contains labeled instances and $\mathcal{D}_U$ contains unlabeled instances.

In the semi-supervised context, the loss function may still involve terms similar to those used in supervised learning, but additional terms are added to leverage the structure of the unlabeled data, such as consistency loss or entropy minimization.

2.1.3 Unsupervised Learning

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. The dataset for unsupervised learning can be denoted as $\mathcal{D} = \{\mathbf{x}_i\}$, with no corresponding output labels y_i.

Clustering is a common technique in unsupervised learning, where the goal is to divide the dataset into groups or clusters. The objective function for clustering, such as the K-means algorithm, tries to minimize the variance within each cluster, which can be represented as:

$$\min_{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_k} \sum_{i=1}^{k} \sum_{\mathbf{x}\in C_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \tag{2.3}$$

where $\boldsymbol{\mu}_i$ is the centroid of cluster C_i, and k is the number of clusters.

2.2 Graph Neural Networks

Graph Neural Networks (GNNs) have emerged as a powerful tool for a wide range of applications where data is inherently structured as graphs, encapsulating relationships among elements. These applications span numerous fields, including but not limited to, modeling

physical systems, learning molecular fingerprints, predicting protein interfaces, and classifying diseases. Such tasks often demand a model capable of processing and learning from graph inputs [22]. GNNs can be generally classified into three categories based on the task: node classification, edge classification, and graph classification. Our experimental work will be concentrated on the node classification task. Here, we introduce several prominent types of GNNs, highlighting their unique approaches to handling graph-structured data.

2.2.1 Graph Convolutional Network

Graph Convolutional Networks (GCNs) represent a class of neural networks specifically designed to operate on graph data, a versatile and widespread data representation across multiple domains [9]. GCNs extend the concept of convolutional neural networks to graph-structured data, thus enabling the model to effectively learn the topological structure of the data.

A graph is typically represented as $G = (V, E)$, where V denotes the set of vertices, and E represents the set of edges. In an undirected graph, the adjacency matrix $A \in \mathbb{R}^{n \times n}$ is symmetric, with n indicating the number of nodes. The degree matrix D is a diagonal matrix where D_{ii} is the degree of node i, which is the sum of the weights of all edges connected to node i.

The foundational layer of a GCN is based on the following propagation rule:

$$H^{(l+1)} = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2.4}$$

Here, $H^{(l)} \in \mathbb{R}^{n \times d_l}$ denotes the activation matrix at the l-th layer, $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ is the corresponding weight matrix, $\hat{A} = A + I_n$ represents the adjacency matrix with added self-loops, and $\hat{D}$ is the degree matrix computed from $\hat{A}$. The function $\sigma(\cdot)$ is a nonlinear activation function, commonly the ReLU function.

The propagation rule in GCNs effectively facilitates the flow of information across the network, as depicted in Figure 2.2. This mechanism allows GCNs to leverage both node features and the overall graph structure. By aggregating feature information from a node's immediate neighbors, GCNs are capable of learning representations that reflect not just the attributes of

individual nodes but also the connectivity patterns within the graph. These representations are instrumental in capturing the essence of the graph's topology and are crucial for tasks such as node classification, link prediction, and graph clustering.

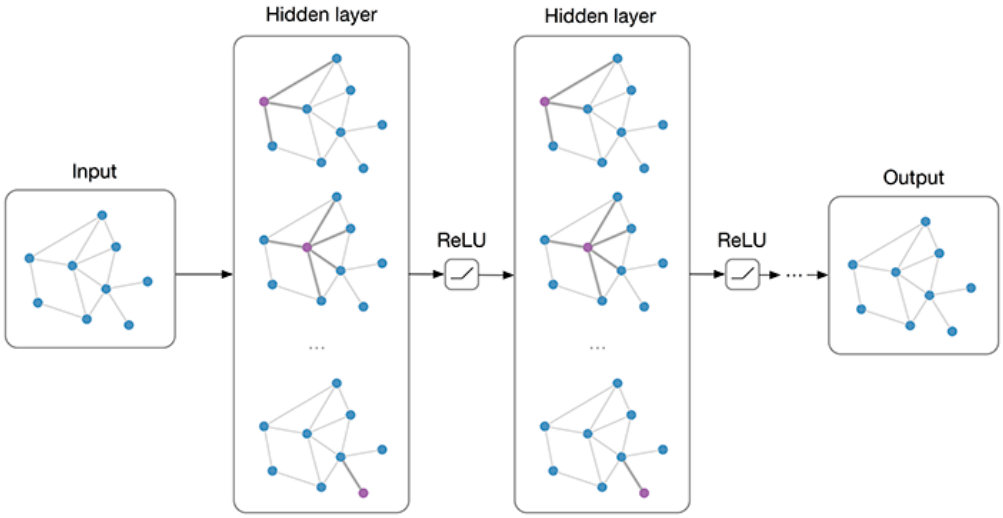

Figure 2.2: Illustration of the GCN structure.

GCNs have been demonstrated to perform effectively on various graph-related tasks, with their ability to capture local neighborhood structures making them particularly suited for node classification tasks. The scalable and efficient nature of GCNs has solidified their role as a cornerstone in graph representation learning [19].

2.2.2 Graph Attention Network

Graph Attention Networks (GATs) introduce an attention mechanism to graph neural networks, enabling the model to assign varying levels of importance to different nodes in the neighborhood [15]. This feature allows GATs to focus more on relevant nodes and less on less relevant ones, adapting to the structure of the graph data dynamically.

Like GCNs, GATs operate on graph-structured data represented as $G = (V, E)$. However, the key difference lies in how node features are aggregated. In GATs, the aggregation process is guided by attention coefficients that indicate the importance of each node's features. The attention mechanism in GATs can be described as follows:

$$\alpha_{ij} = \text{softmax}_j \left(\text{LeakyReLU} \left(\vec{a}^T \left[W\vec{h}_i \parallel W\vec{h}_j \right] \right) \right) \tag{2.5}$$

$$\vec{h}'_i = \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W \vec{h}_j \right) \tag{2.6}$$

Here, $\vec{h}_i$ is the feature vector of node i, W is a shared linear transformation applied to every node, and $\vec{a}$ is a weight vector used in the attention mechanism. The LeakyReLU function introduces non-linearity, and softmax_j normalizes the attention coefficients across all choices of j. The final updated feature $\vec{h}'_i$ for node i is computed as a weighted sum of the features of its neighbors $\mathcal{N}(i)$, with weights given by the attention coefficients α_{ij}.

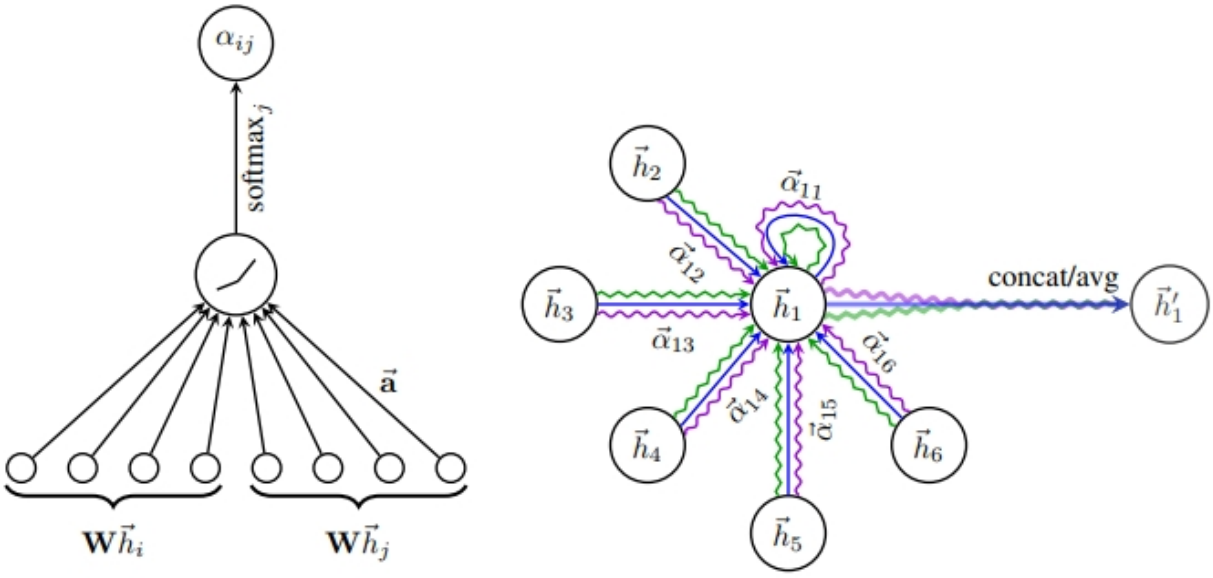

Figure 2.3: Left: The attention mechanism $a(W\tilde{h}_i, W\tilde{h}_j)$ of our model, parametrized by a weight vector $\tilde{a} \in \mathbb{R}^{2F_0}$ and applying a LeakyReLU activation. Right: Illustration of multi-head attention (with K = 3 heads) by node 1 on its neighborhood, showcasing independent attention computations with different styles and colors. The aggregated features from each head are either concatenated or averaged to obtain $\tilde{h}_1^{(0)}$. Cite from the paper Graph Attention Networks[15]

GATs have demonstrated strong performance in tasks where the relative importance of different nodes in the neighborhood varies significantly. The ability to learn different weights

for different parts of the graph makes GATs versatile and powerful for diverse graph-based learning tasks.

The incorporation of attention mechanisms in GATs has expanded the scope and capabilities of graph neural networks, providing a more nuanced approach to learning and representing graph data [15].

2.2.3 Brief Introduction to Various GNNs

A fundamental aspect of Graph Neural Networks (GNNs) is their propagation rules, which can be broadly categorized into non-linear and linear types. These rules govern how information is aggregated and transformed across the network. In this study, our focus has been primarily on non-linear propagation rules. From our perspective, non-linear models represent a more challenging scenario for adversarial attacks due to their complex information processing capabilities.

Table 1: A comparison of propagation rules. Here $\mathbf{X}^{(k)} \in \mathcal{X}$ represents input features after k feature propagation steps and $\mathbf{X}^{(0)} = \mathbf{X}$; $\mathbf{H}^{(k)}$ denotes the hidden features of non-linear GCNs at layer k; $\mathbf{W}$ denotes the weight matrix; σ refers to a activation function; α, β are coefficients.

Method	Type	Propagation rule
GCN [7]	Non-linear	$\mathbf{H}^{(k)} = \sigma\left(\mathbf{S}\mathbf{H}^{(k-1)}\mathbf{W}^{(k-1)}\right)$
APPNP [8]	Non-linear	$\mathbf{H}^{(k)} = (1-\alpha)\mathbf{S}\mathbf{H}^{(k-1)} + \alpha\mathbf{H}^{(0)}$
CGNN [22]	Non-linear	$\mathbf{H}^{(k)} = (1-\alpha)\mathbf{S}\mathbf{H}^{(k-1)}\mathbf{W} + \mathbf{H}^{(0)}$
SGC [21]	Linear	$\mathbf{X}^{(k)} = \mathbf{S}\mathbf{X}^{(k-1)})$
DGC-Euler (ours)	Linear	$\mathbf{X}^{(k)} = (1 - T/K) \cdot \mathbf{X}^{(k-1)} + (T/K) \cdot \mathbf{S}\mathbf{X}^{(k-1)}$

Figure 2.4: A comparative illustration of propagation rules in GNNs [17].

As depicted in Figure 2.4, different GNN architectures employ varied propagation rules. This diversity in approaches highlights the rich landscape of GNN designs, each with its unique strengths and susceptibilities, particularly in the context of adversarial attacks. The non-linear nature of the models we tested adds an additional layer of complexity, both in understanding their behavior and in devising effective strategies to challenge their robustness.

2.3 Natural Language Processing (NLP)

Natural Language Processing (NLP) has been propelled forward by the advent of models like BERT (Bidirectional Encoder Representations from Transformers), which introduces a nuanced approach to language understanding. BERT's fundamental mechanism relies on the Transformer architecture, which uses self-attention mechanisms to process words in relation to all other words in a sentence, diverging from the sequential processing of traditional recurrent neural networks. Pre-trained on a vast corpus of text, BERT captures deep language context through bidirectional training, which is a departure from previous unidirectional models [6].

The essence of BERT lies in its ability to encode rich contextual information for every token in a sentence using the transformer's bidirectional self-attention mechanism, formalized as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.7}$$

Here, Q, K, V represent queries, keys, and values respectively — core components of the attention mechanism, and d_k denotes the dimension of the keys. Through such mechanisms, BERT achieves remarkable performance across diverse NLP tasks, setting new standards for machine understanding of language [14].

2.4 Adversarial Machine Learning

Adversarial Machine Learning is a research field that resides at the intersection of machine learning and cybersecurity. It is primarily concerned with the vulnerability of machine learning systems to adversarial examples – input data instances that are deliberately engineered to cause the model to make incorrect predictions or decisions. These adversarial examples are typically created by adding small, often imperceptible perturbations to the input data that lead to significant changes in the output [2].

2.4.1 Brief Introduction to AML

Formally, given a machine learning model $f : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space, an adversarial example $\mathbf{x}_{adv}$ can be defined with respect to an original input $\mathbf{x} \in \mathcal{X}$ and its true label $y \in \mathcal{Y}$ as:

$$\mathbf{x}_{adv} = \mathbf{x} + \delta, \text{ such that } f(\mathbf{x}_{adv}) \neq y \tag{2.8}$$

The perturbation δ is usually constrained to be small under some p-norm, commonly the L_∞ norm or L_2 norm, to ensure the adversarial example is similar to the original input:

$$\|\delta\|_p \leq \epsilon \tag{2.9}$$

where ϵ is a small constant that bounds the magnitude of the perturbation.

Adversarial attacks can be formulated as optimization problems, where the objective is to maximize the loss of the machine learning model on the adversarial example, subject to δ's magnitude constraints. For a model with loss function $\mathcal{L}(f(\mathbf{x}), y)$, an adversarial attack seeks to solve:

$$\max_{\delta} \mathcal{L}(f(\mathbf{x} + \delta), y) \quad \text{subject to} \quad \|\delta\|_p \leq \epsilon \tag{2.10}$$

This type of attack is often called an L_p-norm attack, where p can take values such as 0, 2, or ∞ to denote different norms.

The study of adversarial machine learning also includes the development of defensive strategies, such as adversarial training, where the model is trained on a mixture of adversarial and clean examples, and robust optimization, which aims to minimize the worst-case loss over perturbations:

$$\min_{\theta} \max_{\|\delta\|_p \leq \epsilon} \mathcal{L}(f_\theta(\mathbf{x} + \delta), y) \tag{2.11}$$

Here, θ represents the parameters of the model f, and the inner maximization problem represents the adversarial attack during the training process.

In essence, adversarial machine learning helps to understand the limits of current machine learning algorithms and guides the design of more robust models capable of withstanding adversarial perturbations, which is crucial for security-sensitive applications [10].

2.4.2 Categories of AML

Researchers have classified attacks on machine learning systems in several ways. As outlined in "Adversarial Machine Learning" [16], attacks are typically divided into three primary categories based on timing, information access, and objectives.

Table 2.1: Three dimensions of attacks on machine learning.

Attack Timing	Decision time (e.g., evasion attacks) vs. training time (e.g., poisoning attacks)
Attacker Information	White-box (full knowledge) vs. black-box (limited knowledge) attacks
Attack Goals	Targeted attacks (specific misclassification) vs. integrity attacks (general reliability disruption)

In our experimental framework, the categorization of adversarial attacks is predominantly founded on the timing aspect. Understanding when an attack occurs is pivotal in characterizing its nature and formulating defensive strategies. Two primary types of adversarial attacks, distinguished by their timing, are central to our exploration:

- **Decision-time Attacks:** Also known as evasion attacks, these are executed after the training of the machine learning model. The attacker, at this juncture, manipulates the input data to the well-established model with the intention of eliciting incorrect predictions. This type of attack exploits the model's decision-making at inference time, aiming to bypass or deceive the model without altering its underlying structure.
- **Poisoning Attacks:** These are carried out during the model's training phase. Here, the attacker injects specially crafted data into the training set, which is then used to train the model. The corrupted training process results in a model that is compromised and whose accuracy is deliberately reduced once it is deployed for prediction.

Each attack type targets a different phase in the lifecycle of a model, with distinct methodologies and implications. Decision-time attacks manipulate the model's ability to generalize from learned patterns, while poisoning attacks directly affect the learning process by contaminating the source of knowledge—the training data. The delineation between these attack types is critical to developing robust defensive mechanisms and is visually summarized in Figure 2.5.

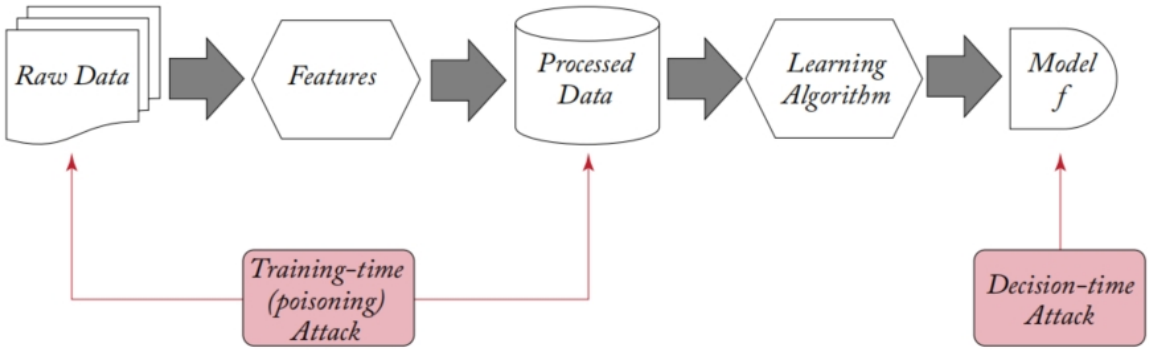

Figure 2.5: Illustration of the timing of attacks in adversarial machine learning, contrasting decision-time (evasion) attacks against poisoning (training-phase) attacks, as cited from "Adversarial Machine Learning" [16].

2.4.3 Adversarial Attacks on Graph

The intersection of adversarial learning and GNN is an intriguing space where a model (such as GCN, GAT) and its antagonist engage in a strategic duel. This involves the adversary artfully tweaking the graph structure or node attributes in a bid to manipulate the model's predictions [24].

One pioneering approach to disrupting GNNs is the NETTACK [24] method, which targets the structural integrity of the graph to mount poisoning attacks during the training phase of a GCN model. Alternatively, RL-S2V [5] leverages reinforcement learning to execute evasion attacks at testing time, aiming to bypass the model's defenses by altering graph data in real time. Methods in references [4] and [18] take a different tack by utilizing gradient information for more precise poisoning attacks. Reference [4] employs an iterative process to manipulate node connections with significant gradient values, effectively attacking the graph

in its embedding space, while [18] uses integrated gradients to approximate the model's gradients and flips binary values to perturb the graph data strategically [5, 25].

As GNNs continue to gain traction in sensitive areas like social network analysis, cybersecurity, and bioinformatics, the stakes for protecting against adversarial attacks are heightened. The potential for adversaries to exploit these models underscores an urgent need for ongoing research into the nature of adversarial learning in graph data, aiming to fortify models against such insidious attacks [13].

Chapter 3

Methodology

This chapter delineates the methodologies adopted and the experimental setup designed to assess the efficacy of the proposed adversarial attacks on graph neural networks (GNNs) based on text datasets.

3.1 Workflow

The workflow is crucial for illustrating the step-by-step process we employ to conceptualize, implement, and evaluate adversarial attacks. To ensure clarity and facilitate understanding, we depict our research workflow, as shown in Figure 3.1.

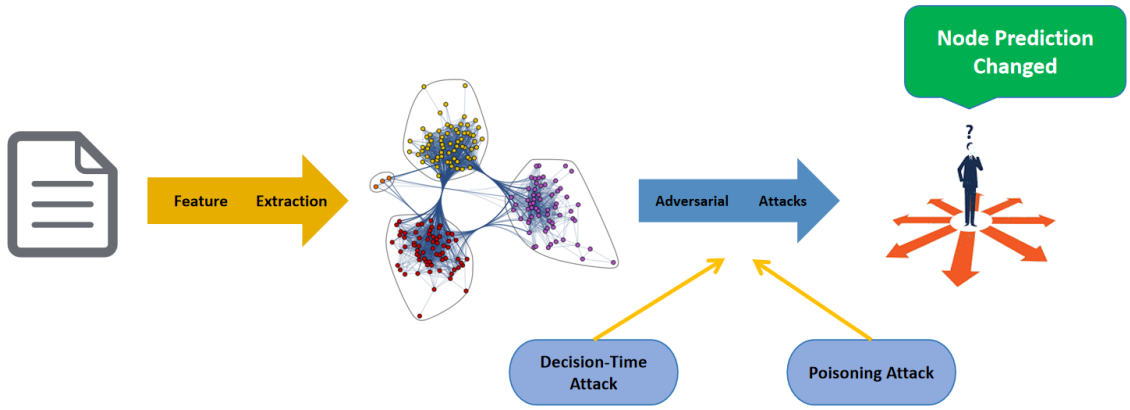

Figure 3.1: Schematic representation of the research workflow, outlining the progression from text data processing to adversarial attack evaluation.

The workflow begins with our textual dataset, from which we extract features using a pre-trained model - BERT. The extracted features form the foundation of our graph construction, where textual instances are transformed into graph nodes, and their relationships are depicted as edges.

Upon establishing the graph structure, we conduct two types of adversarial attacks: decision-time and poisoning attacks. Decision-time attacks are executed at the moment of inference, aiming to mislead the model's predictions on the fly. Conversely, poisoning attacks occur during the model's training phase, where the data is manipulated to compromise the model's learning process permanently.

Finally, the effectiveness of each attack is measured by evaluating the changes in the model's predictions for specific nodes. The degree to which the attacks can alter these predictions serves as a testament to their potency. By comparing the model's performance before and after the attacks, we gain insights into the robustness of GNNs against adversarial interventions and the necessity for developing more resilient graph learning algorithms.

3.2 Data Preprocessing

3.2.1 Graph Building by Using Text Dataset

To facilitate our adversarial studies on graph-based models, we preprocess the textual data obtained from the Hellaswag dataset [21], which is publicly accessible at https://huggingface.co/datasets/hellaswag. We employ the BERT[6] model to derive rich contextual representations from the 'ctx_a' textual features within the dataset. These representations are further processed to serve as node embeddings within our graph.

In constructing the graph, we introduce edges by assessing the relationship between node embeddings. This is achieved by calculating the cosine similarity between every pair of node embeddings, thereby quantifying the strength of their relationships. An edge is formed between two nodes if their cosine similarity exceeds a predefined threshold, ensuring that only significantly related nodes are connected.

We then process the labels through a Label Encoder to facilitate their incorporation into a non-directed graph structure. The encoder transforms categorical labels into a numerical format that is amenable to the algorithms employed in later stages.

It is crucial for our experimental validity to maintain a balanced distribution of labels across the graph. This balance ensures that our adversarial attacks are not inadvertently biased towards any particular label, which could skew the results and impair the interpretability of our findings.

The cosine similarity, a fundamental measure in the graph construction, is defined as follows:

$$\text{cosine similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{3.1}$$

where:

- $\mathbf{A} \cdot \mathbf{B}$ denotes the dot product between the feature vectors $\mathbf{A}$ and $\mathbf{B}$,
- $\|\mathbf{A}\|$ represents the Euclidean norm of vector $\mathbf{A}$, computed as $\sqrt{\mathbf{A} \cdot \mathbf{A}}$,
- Similarly, $\|\mathbf{B}\|$ represents the Euclidean norm of vector $\mathbf{B}$, computed as $\sqrt{\mathbf{B} \cdot \mathbf{B}}$.

3.2.2 Utilizing Established Graph Datasets

To validate the effectiveness of our adversarial attack methodologies, we extend our experiments to include established graph datasets, such as Cora and Reddit. These datasets are pivotal in the field of graph neural networks, often serving as benchmarks for assessing the node classification tasks.

Cora Dataset

The Cora dataset comprises a citation network where nodes are scientific papers, and edges represent citations. Each paper is associated with a binary feature vector that encodes the presence of certain words, and papers are classified into one of seven classes based on their

topics. This dataset is instrumental in evaluating the performance of graph-based models in citation networks and topic categorization.

CiteSeer Dataset

Similar to the Cora dataset, CiteSeer is also a citation network. In this dataset, nodes represent academic papers, and edges indicate citation links between these papers. Each paper is described by a textual feature vector, representing the words contained within the paper using a bag-of-words approach. The papers are categorized into one of six classes, corresponding to different research areas. The CiteSeer dataset challenges models to accurately classify papers into these categories based on both their content and citation context, making it an excellent testbed for assessing the resilience of graph neural networks against adversarial attacks in real-world scenarios.

3.3 Adversarial Attacks

3.3.1 Decision-time Attack

The concept of decision-time attacks, where adversarial examples are introduced during a model's inference phase, has been significantly advanced by two techniques: the Fast Gradient Sign Method (FGSM) proposed by Goodfellow et al. in "Explaining and Harnessing Adversarial Examples" [7], and the Projected Gradient Descent (PGD) method [10].

FGSM[7] crafts adversarial examples by applying a perturbation η that maximizes the neural network's loss within a given ∞-norm constraint ϵ. The perturbation is defined as:

$$\eta = \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y)) \tag{3.2}$$

Conversely, PGD is an iterative attack method that takes multiple small steps in the direction of the gradient, followed by a projection step to ensure that the perturbations stay within the ϵ-ball. This method is generally considered more powerful than FGSM, as it explores a

wider space for potential adversarial examples. The PGD update rule can be formulated as:

$$x^{(t+1)} = \text{Proj}_{x+S}\left(x^{(t)} + \alpha \cdot \text{sign}(\nabla_x J(\theta, x^{(t)}, y))\right) \tag{3.3}$$

where Π denotes the projection operation, and α is the step size.

Figure 3.2 below provides a visualization of how adversarial examples can be generated using these methods.

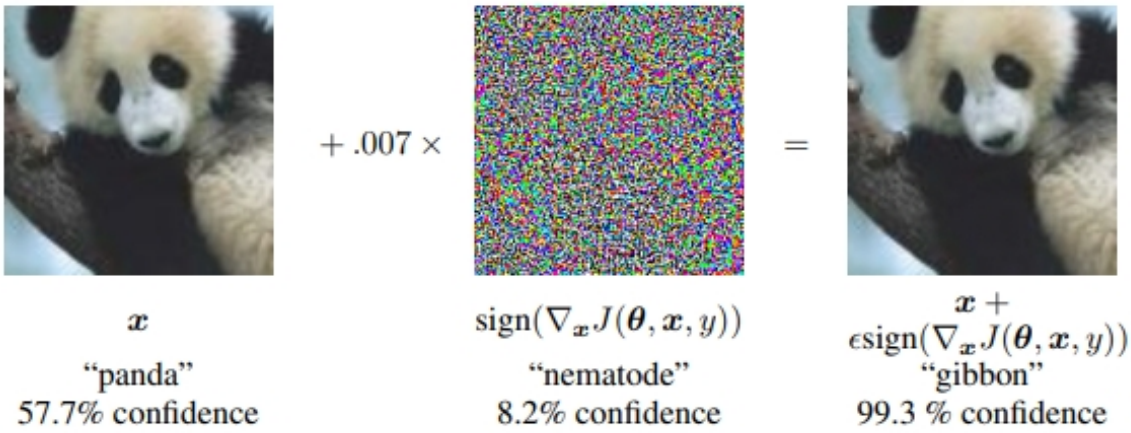

Figure 3.2: A demonstration of fast adversarial example generation applied to GoogLeNet on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. In this instance, ϵ of 0.007 corresponds to the magnitude of the smallest bit of an 8-bit image encoding after GoogLeNet's conversion to real numbers.[10]

Both these techniques are employed in our study to assess the vulnerability of Graph Neural Networks (GNNs) against adversarial interference, underscoring the need for robust models. The effectiveness of each method under different norm constraints (1-norm, 2-norm, and ∞-norm) will be investigated to determine their impact on the success rate of adversarial attacks.

Decision-time Attack with Top-K Degree Nodes

In addition to employing FGSM and PGD techniques for decision-time attacks, our experiments specifically focus on targeting the Top-K degree nodes within the graph. By directing

the adversarial methods towards these high-degree nodes, we aim to evaluate the impact of attacks on the most influential nodes in the network, which could potentially have a larger effect on the overall model performance.

The experimental procedure for the Top-K degree decision-time attack is outlined as follows:

Algorithm 1 Pseudocode for Top-K Degree Decision-Time Attack

0: **Input:** Graph $G(V, E)$, Node embeddings Emb, Top-K nodes K
0: **Output:** Poisoned graph
0: Initialize adversarial perturbation $\eta \leftarrow \mathbf{0}$
0: **for** each node $v \in K$ **do**
0: Apply PGD to generate perturbation η_v
0: Update node embedding $Emb(v) \leftarrow Emb(v) + \eta_v$
0: **end for**
0: Apply modified embeddings to GNN model
0: Evaluate model performance on poisoned graph
0: **return** Modified Graph =0

This pseudocode details the steps to apply decision-time attacks using either FGSM or PGD on the top degree nodes of the graph. The goal is to observe how the model's performance is affected when the most connected nodes in the graph are perturbed, providing insights into the robustness of GNNs under targeted adversarial conditions.

Figure 3.2 demonstrates the concept of adversarial example generation, which is fundamental to our decision-time attack strategy.

3.3.2 Poisoning Attack

Poisoning attacks are insidious strategies where adversaries deliberately manipulate training data to skew the model's learning process. As discussed in earlier chapters, these attacks subtly inject 'poison' into the dataset, crafting inputs that, once trained upon, cause the model to make incorrect predictions. Our study delves into the design of algorithms capable of such data manipulation, simulating how these poisoned inputs can mislead a GNN model. The impact of these attacks on the model's predictive accuracy will be thoroughly

investigated, providing insights into the vulnerability of GNNs and the necessity for robust defensive mechanisms.

Applying Gaussian Random Noises to Poison Node Embeddings

In an attempt to simulate adversarial conditions, Gaussian random noise is added to the node embeddings to subtly corrupt the input data. This technique, widely used in image processing adversarial attacks, is adapted to the context of graph data as follows:

Let $\mathbf{E} = \{\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n\}$ be the set of node embeddings obtained from the graph where $\mathbf{e}_i \in \mathbb{R}^d$ represents the embedding of the i-th node. We perturb each embedding $\mathbf{e}_i$ by adding Gaussian random noise $\eta_i \sim \mathcal{N}(\mu, \sigma^2 I)$, where μ and σ are the mean and standard deviation of the Gaussian distribution, and I is the identity matrix. The perturbed embedding $\tilde{\mathbf{e}}_i$ is given by:

$$\tilde{\mathbf{e}}_i = \mathbf{e}_i + \eta_i \tag{3.4}$$

The noise injection process can be fine-tuned by adjusting μ and σ, enabling an exploration of the graph neural network's robustness across varying degrees of adversarial perturbation. The objective is to observe how such disturbances affect the model's learning and generalization capabilities, thereby assessing its vulnerability to adversarial attacks.

Deviating the Distribution of Node Embeddings

A critical aspect of this poisoning strategy involves deviating the node embeddings from their typical distribution. We achieve this by calculating the mean of the embeddings and using a control parameter *lambda* to manipulate the degree to which embeddings are altered towards this mean. Specifically, for each node embedding $\tilde{\mathbf{e}}_i$, we adjust it in relation to the mean embedding $\bar{\mathbf{e}}$ as follows:

$$\tilde{\mathbf{e}}_i = (1 - \lambda)\tilde{\mathbf{e}}_i + \lambda\bar{\mathbf{e}} \tag{3.5}$$

where $\bar{\mathbf{e}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{e}_i$ is the mean of the original embeddings. This process systematically skews the embeddings towards a more uniform distribution, potentially making the model less sensitive to specific graph structures and thereby reducing its overall accuracy and effectiveness.

Applying Graph Contrastive Learning to Poison Node Embeddings

Graph Contrastive Learning (GCL) can be exploited for poisoning attacks aimed at compromising a Graph Neural Network's (GNN) performance [23]. We cast this as an optimization problem targeting the node embeddings. The adversarial objective is to find perturbations that, when applied to the original node embeddings, hinder the GNN's accuracy. The optimization problem is given by:

$$\begin{aligned} \underset{E(x^*)}{\text{minimize}} \quad & L = \lambda_1 L_{sim} + \lambda_2 L_{dis} \\ \text{subject to} \quad & E(x^*) = E(x) + \delta, \\ & \|\delta\|_p \leq \epsilon, \\ & x + \delta \in \mathcal{X}, \end{aligned} \tag{3.6}$$

where L_{sim} is the similarity loss between poisoned embeddings, L_{dis} is the dissimilarity loss encouraging differentiation from original embeddings, δ is the adversarial perturbation bounded by ϵ under norm p, and $\mathcal{X}$ represents the permissible embedding space.

The similarity loss L_{sim} is defined as the negative mean similarity between all poisoned embeddings:

$$L_{sim} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\text{sim}(E(x_i^*), E(x_j^*)), \tag{3.7}$$

and the dissimilarity loss L_{dis} is the log-sum-exp of similarities between original and poisoned embeddings, promoting divergence:

$$L_{dis} = \sum_{i=1}^{n}\log\left(\sum_{j=1}^{n} e^{\text{sim}(E(x_i), E(x_j^*))}\right). \tag{3.8}$$

In this formulation, λ_1 and λ_2 are hyperparameters that balance the trade-off between similarity and dissimilarity losses, thus guiding the severity of the poisoning.

Applying this framework, we specifically target the Top-K degree nodes within the graph, exploiting their influential roles in the network's structure. By perturbing these nodes, we aim to induce significant impacts on the GNN's functionality.

Algorithm 2 Pseudocode for Poisoning Node Embeddings using GCL

0: **Input:** Graph $G(V, E)$, Node embeddings Emb, Top-K nodes K
0: **Output:** Poisoned embeddings Emb^*
0: Initialize perturbation $\delta \leftarrow \mathbf{0}$
0: **for** each node $v \in K$ **do**
0: Calculate L_{sim} and L_{dis} for v
0: Update δ_v using the optimization problem
0: $Emb^*(v) \leftarrow Emb(v) + \delta_v$
0: Enforce $\|\delta_v\|_p \leq \epsilon$
0: **end for**
0: **return** Emb^* =0

This pseudocode outlines the procedure for applying GCL to poison the embeddings of the Top-K degree nodes. The approach involves iteratively updating the perturbation δ to solve the defined optimization problem, ensuring the perturbed embeddings $E(x^*)$ deviate the GNN's predictions.

Chapter 4

Experiment Settings & Results

This chapter presents the results of deploying models on both pure datasets and those subjected to adversarial attacks, including both decision-time and poisoning attacks. The performance of two prominent Graph Neural Network architectures, GCN and GAT, is evaluated to assess their robustness and efficacy.

4.1 Model Settings

4.1.1 GCN settings

For the Graph Convolutional Network (GCN) setup [9], we have designed a model that encompasses three convolutional layers, each employing Rectified Linear Unit (ReLU) as the activation function. This model is specifically configured to tackle multi-class classification problems. We utilize the cross-entropy loss function, which is standard for such classification tasks and particularly suited to the datasets we are working with. To counter the risk of overfitting and to bolster the model's generalization capabilities, dropout layers are strategically incorporated into the network's architecture. The rationale behind this design is to capitalize on the inherent capabilities of GCNs in processing and interpreting graph-structured data, thereby achieving effective and accurate classification results. Our goal is to harness and demonstrate the strengths of GCNs in handling complex patterns inherent in graph data for reliable classification outcomes.

4.1.2 GAT settings

The Graph Attention Network (GAT)[15] employs attention mechanisms to weigh the influence of neighboring nodes during the feature aggregation phase. Our GAT model consists of three attention layers, each followed by a ReLU activation. Similar to the GCN setup, dropout layers are used to reduce overfitting. The attention mechanism in GAT allows the model to focus on the most relevant parts of the graph structure, potentially improving performance over the standard GCN in tasks involving complex node interrelations.

Both models are trained and evaluated on the pure datasets to establish a performance benchmark. The results from these experiments will provide a baseline against which the impact of adversarial attacks can be measured.

4.2 Pure Dataset

We begin by evaluating the performance of our models on pure datasets, free from any adversarial modifications. The datasets used for this assessment include the text-based Hellaswag dataset and the graph datasets Cora and CiteSeer. This evaluation serves as a baseline to understand the models' behavior under normal conditions.

Table 4.1: Performance comparison of models on multiple datasets.

Dataset	**Model**	**Validation Acc**	**Testing Acc**
Hellaswag	GCN	93.75%	92.42%
Hellaswag	GAT	92.19%	93.94%
Cora	GCN	92.19%	93.94%
Cora	GAT	77.00%	79.00%
CiteSeer	GCN	64.60%	65.30%
CiteSeer	GAT	64.80%	65.80%

4.3 Decision-time Attack

In our decision-time attack experiments, we utilized Projected Gradient Descent (PGD) with various norms (1-norm, 2-norm, and ∞-norm) within an epsilon-ball to bound the changes

in the dataset. The impact of these attacks on the final accuracy rate can be influenced by two main factors: the choice of norm and the value of ϵ. A smaller norm restricts the perturbation boundary, resulting in less perturbation, while an increase in ϵ allows for greater perturbation. Below, we present the results showing how the accuracy rate varies with different $\epsilon = 0.1$ values for both the Hellaswag and Cora datasets:

$$\eta = \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y)) \tag{4.1}$$

Table 4.2: Test Loss and Accuracy for Different Norms with $\epsilon = 0.1$ on Hellaswag and Cora Datasets

Dataset	**Norm**	**Test Loss**	**Test Accuracy**
Hellaswag	L_1	0.1223	95.4545%
Hellaswag	L_2	0.3755	83.3333%
Hellaswag	L_∞	1.3701	30.3030%
Cora	L_1	0.7637	77.3%
Cora	L_2	12.7102	1.0%
Cora	L_∞	304.5575	0.9%

After examining the impact of different norms, we further explored the influence of varying epsilon (ϵ) values. As ϵ increases, test accuracy initially drops significantly, reflecting a typical response to adversarial examples. However, beyond a certain point, the accuracy tends to stabilize, suggesting an inherent level of robustness in the models against adversarial perturbations.

The numerical results for different epsilon ϵ values are summarized in the following table 4.3:

4.4 Poisoning Attack

4.4.1 Net Attack

The Net Attack strategically perturbs both the feature and structure of a graph. It employs the constraint $|A^{(t)} - A^{(0)}| + |X^{(t)} - X^{(0)}| < \Delta$ to guide the update process, ensuring the perturbations are subtle yet effective.

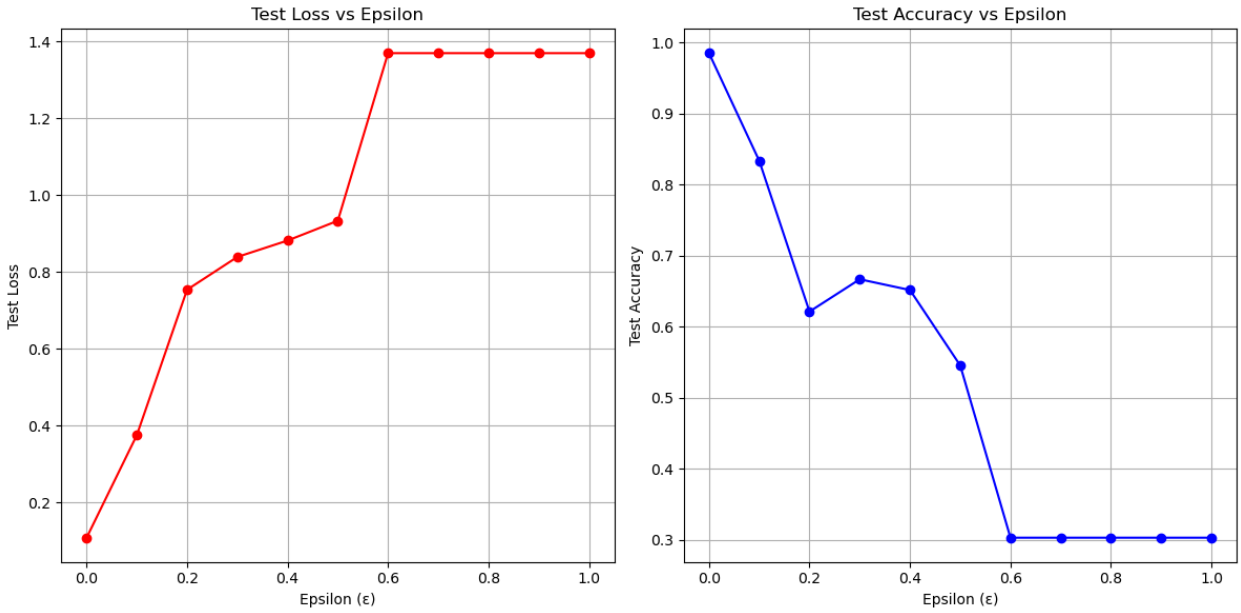

Figure 4.1: Impact of varying epsilon ϵ in PGD attack on the Hellaswag dataset using the L_2 norm.

In our experiments, the Net Attack methodology, as described in [24], is employed to evaluate its effectiveness in conducting poisoning attacks. This approach is particularly notable for its dual focus on both graph features and structure, providing a comprehensive view of its impact on model performance.

Our results indicate varying levels of impact based on the type of perturbation applied. Isolated feature perturbations show a relatively modest effect on model performance, as seen in Figure 4.3. This observation suggests that feature manipulation alone may not sufficiently compromise a robust graph model.

In contrast, the application of both feature and structural perturbations results in a more pronounced degradation of model performance, as demonstrated in Figure 4.4. This highlights the effectiveness of combined feature and structural alterations in poisoning attacks.

These findings underscore the need for considering both feature and structural elements in designing effective poisoning strategies for graph neural networks. Additionally, they shed light on potential vulnerabilities in these models, offering insights for enhancing their resilience against adversarial manipulations.

Table 4.3: Test Loss and Accuracy for Different Epsilon ϵ Values using L_2 Norm

Epsilon ϵ	**Test Loss**	**Accuracy**
0.0	0.1083	98.4848%
0.1	0.3755	83.3333%
0.2	0.7541	62.1212%
0.3	0.8394	66.6667%
0.4	0.8821	65.1515%
0.5	0.9336	54.5455%
0.6	1.3701	30.3030%
0.7	1.3701	30.3030%
0.8	1.3700	30.3030%
0.9	1.3700	30.3030%
1.0	1.3700	30.3030%

4.4.2 Meta Attack

The Meta Attack algorithm [25] conceptualizes the graph structure matrix as a hyperparameter. The key process involves computing the gradient of the attacker's loss with respect to this matrix after training, denoted as $\nabla_{G^{\text{meta}}}$. This is mathematically represented as:

$$\nabla_{G^{\text{meta}}} := \nabla_G L_{\text{atk}}(f_{\theta^*}(G)),$$

subject to the condition that $\theta^* = \text{opt}_\theta(L_{\text{train}}(f_\theta(G)))$, where L_{train} and L_{atk} are the training and attack loss functions, respectively.

The attack is approximated through the following equation:

$$\nabla_{A^{\text{meta}}} \approx \sum_{t=1}^{T} \lambda \nabla_A L_{\text{train}}(f_{\tilde{\theta}_t}(A; X)) + (1-\lambda)\nabla_A L_{\text{self}}(f_{\tilde{\theta}_t}(A; X)).$$

This method combines training loss and self-training loss, modulated by the parameter λ, to effectively simulate a poisoning attack on GNNs. It highlights the criticality of the graph structure in the resilience of GNNs against sophisticated adversarial techniques.

Interestingly, our experiments with the Meta Attack on the same text dataset used for the Net Attack yielded similar results. Contrary to our expectations, the Meta Attack did not significantly perturb the dataset. More surprisingly, it appeared to increase the accuracy

Algorithm 1: NETTACK: Adversarial attacks on graphs

Input: Graph $G^{(0)} \leftarrow (A^{(0)}, X^{(0)})$, target node v_0,
attacker nodes $\mathcal{A}$, modification budget Δ

Output: Modified Graph $G' = (A', X')$

Train surrogate model on $G^{(0)}$ to obtain W // Eq. (13);

$t \leftarrow 0$;

while $|A^{(t)} - A^{(0)}| + |X^{(t)} - X^{(0)}| < \Delta$ **do**

$C_{struct} \leftarrow \texttt{candidate\_edge\_perturbations}(A^{(t)}, \mathcal{A})$;

$e^* = (u^*, v^*) \leftarrow \underset{e \in C_{struct}}{\arg\max}\ s_{struct}\left(e; G^{(t)}, v_0\right)$;

$C_{feat} \leftarrow \texttt{candidate\_feature\_perturbations}(X^{(t)}, \mathcal{A})$;

$f^* = (u^*, i^*) \leftarrow \underset{f \in C_{feat}}{\arg\max}\ s_{feat}\left(f; G^{(t)}, v_0\right)$;

if $s_{struct}(e^*; G^{(t)}, v_0) > s_{feat}(f^*; G^{(t)}, v_0)$ **then**

$G^{(t+1)} \leftarrow G^{(t)} \pm e^*$;

else $G^{(t+1)} \leftarrow G^{(t)} \pm f^*$;

$t \leftarrow t + 1$;

return $: G^{(t)}$

// Train final graph model on the corrupted graph $G^{(t)}$;

Figure 4.2: Net Attack Pseudocode.

rate, an outcome that deviates from the anticipated direction of impact. This unexpected result prompts further investigation into the underlying reasons why this attack strategy might inadvertently enhance model performance, rather than degrade it.

4.4.3 Adding Random Noises Poisoning Attack

New added part...

4.4.4 Mean Node Embedding Poisoning Attack

In this experiment, we focused exclusively on node embeddings. Implementing the algorithm described in chapter 3.3.2 "Deviating the Distribution of Node Embeddings," we adjusted

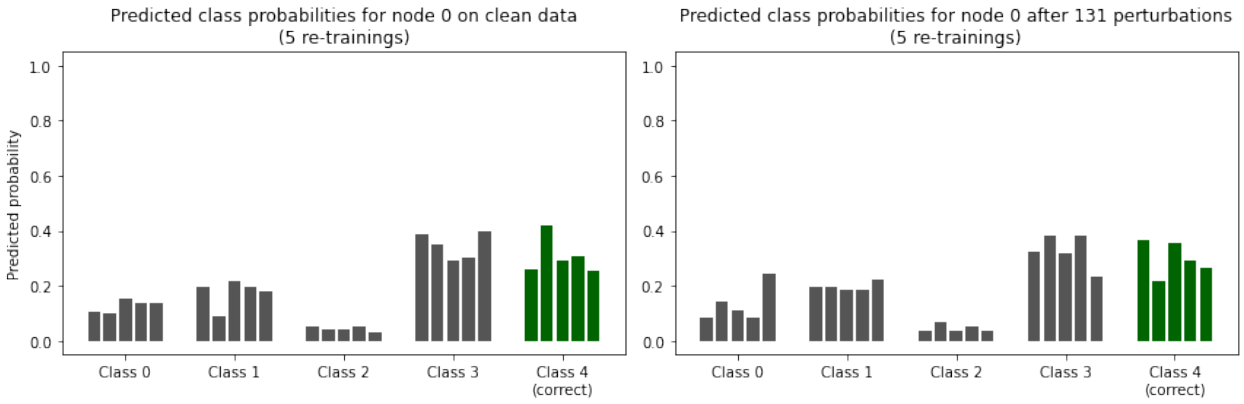

Figure 4.3: Results for Net Attack when only perturbing features.

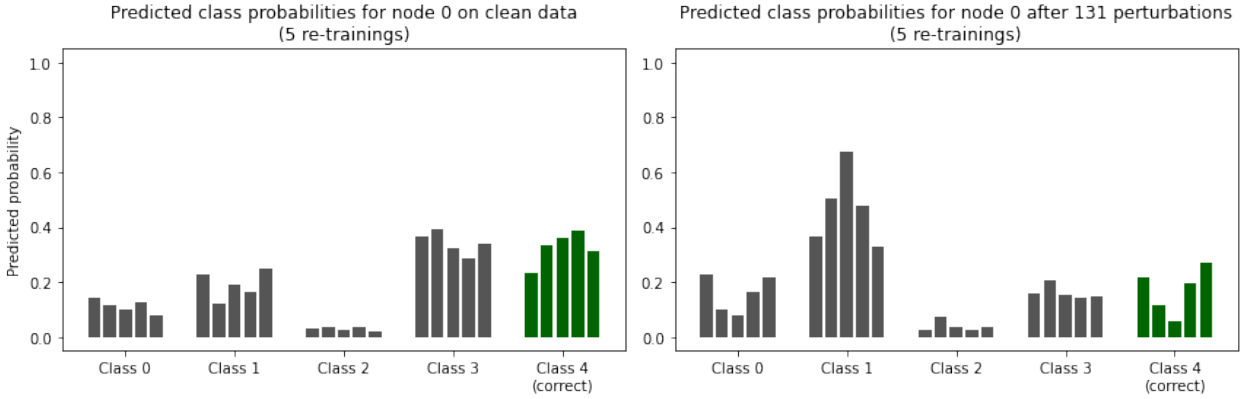

Figure 4.4: Results for Net Attack when perturbing both features and structure.

the lambda parameter to control the degree of deviation. The experiment was conducted on the Cora dataset, with the results depicted in Figure 4.8 and further detailed in the table below.

The numerical results, showcasing the test loss and accuracy for various lambda values, are summarized in the following table:

These results indicate how varying the lambda parameter affects the effectiveness of the poisoning attack, with different values leading to significant variations in test loss and accuracy.

Algorithm 1: Poisoning attack on graph neural networks with meta gradients and self-training

Input: Graph $G = (A, X)$, modification budget Δ, number of training iterations T, training class labels C_L

Output: Modified graph $\hat{G} = (\hat{A}, X)$

$\hat{\theta} \leftarrow$ train surrogate model on the input graph using known labels C_L;

$\hat{C}_U \leftarrow$ predict labels of unlabeled nodes using $\hat{\theta}$;

$\hat{A} \leftarrow A$;

while $\|\hat{A} - A\|_0 < 2\Delta$ **do**

  randomly initialize θ_0;

  for t *in* $0 \dots T-1$ **do**

    $\theta_{t+1} \leftarrow \text{step}\left(\theta_t, \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{A}, X)); C_L\right)$; `// update e.g. via gradient descent`

  `// Compute meta gradient via backprop through the training procedure`

  $\nabla^{\text{meta}}_{\hat{A}} \leftarrow \nabla_{\hat{A}} \mathcal{L}_{\text{self}}(f_{\theta_T}(\hat{A}, X); \hat{C}_U)$;

  $S \leftarrow \nabla^{\text{meta}}_{\hat{A}} \odot (-2\hat{A} + 1)$; `// Flip gradient sign of node pairs with edge`

  $e' \leftarrow$ maximum entry (u, v) in S that fulfills constraints $\Phi(G)$;

  $\hat{A} \leftarrow$ insert or remove edge e' to/from $\hat{A}$;

$\hat{G} \leftarrow (\hat{A}, X)$;

return : $\hat{G}$

Figure 4.5: Poisoning attack on graph neural networks with meta gradients and self-training.

4.4.5 GCL Poisoning Attack

In Chapter 3.3.2, we introduced an algorithm employing Graph Contrastive Learning for executing poisoning attacks on graph-based models. In this subsection, we present the results of these attacks.

As depicted in Figure 4.9, there is a noticeable decline in the model's performance when subjected to the poisoned dataset, with accuracy dropping to approximately 42%. To prevent overfitting and ensure the validity of our results, early stopping was implemented based on the validation set's performance. This substantial decrease in accuracy underscores the effectiveness of the Graph Contrastive Learning approach in compromising the model, thereby highlighting potential vulnerabilities in graph neural networks when exposed to such adversarial techniques.

Algorithm 2: Poisoning attack on GNNs with approximate meta gradients and self-training

Input: Graph $G = (A, X)$, modification budget Δ, number of training iterations T, gradient weighting λ, training class labels C_L

Output: Modified graph $\hat{G} = (\hat{A}, X)$

$\hat{\theta} \leftarrow$ train surrogate model on the input graph using known labels C_L;

$\hat{C}_U \leftarrow$ predict labels of unlabeled nodes using $\hat{\theta}$;

$\hat{A} \leftarrow A$;

while $\|\hat{A} - A\|_0 < 2\Delta$ **do**

  randomly initialize θ_0;

  $\nabla^{\text{meta}}_{\hat{A}} \leftarrow \lambda \nabla_{\hat{A}} \mathcal{L}_{\text{train}}(f_{\theta_0}(\hat{A}; X); C_L) + (1-\lambda) \nabla_{\hat{A}} \mathcal{L}_{\text{self}}(f_{\theta_0}(\hat{A}; X); \hat{C}_U)$

  for t *in* $0 \dots T-1$ **do**

    $\theta_{t+1} \leftarrow \text{step}\,(\theta_t, \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{A}, X)); C_L)$; `// update e.g. via gradient descent`

    $\tilde{\theta}_{t+1} \leftarrow \text{stop\_gradient}(\theta_{t+1})$; `// no backprop through training`

    $\nabla^{\text{meta}}_{\hat{A}} \leftarrow \nabla^{\text{meta}}_{\hat{A}} + \lambda \nabla_{\hat{A}} \mathcal{L}_{\text{train}}(f_{\tilde{\theta}_{t+1}}(\hat{A}; X); C_L) + (1-\lambda) \nabla_{\hat{A}} \mathcal{L}_{\text{self}}(f_{\tilde{\theta}_{t+1}}(\hat{A}; X); \hat{C}_U)$

  $S \leftarrow \nabla^{\text{meta}}_{\hat{A}} \odot (-2\hat{A} + 1)$; `// Flip gradient sign of node pairs with edge`

  $e' \leftarrow$ maximum entry (u, v) in S that fulfills constraints $\Phi(G)$;

  $\hat{A} \leftarrow$ insert or remove edge e' to/from $\hat{A}$;

$\hat{G} \leftarrow (\hat{A}, X)$;

return : $\hat{G}$

Figure 4.6: Poisoning attack on GNNs with approximate meta gradients and self-training.

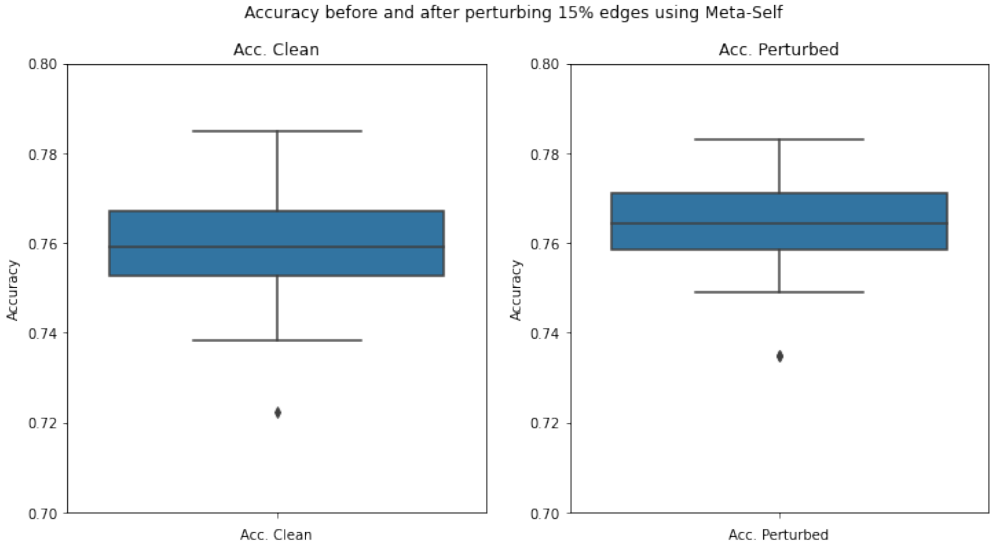

Figure 4.7: Meta Attack Result applied on the text dataset "Hellaswag".

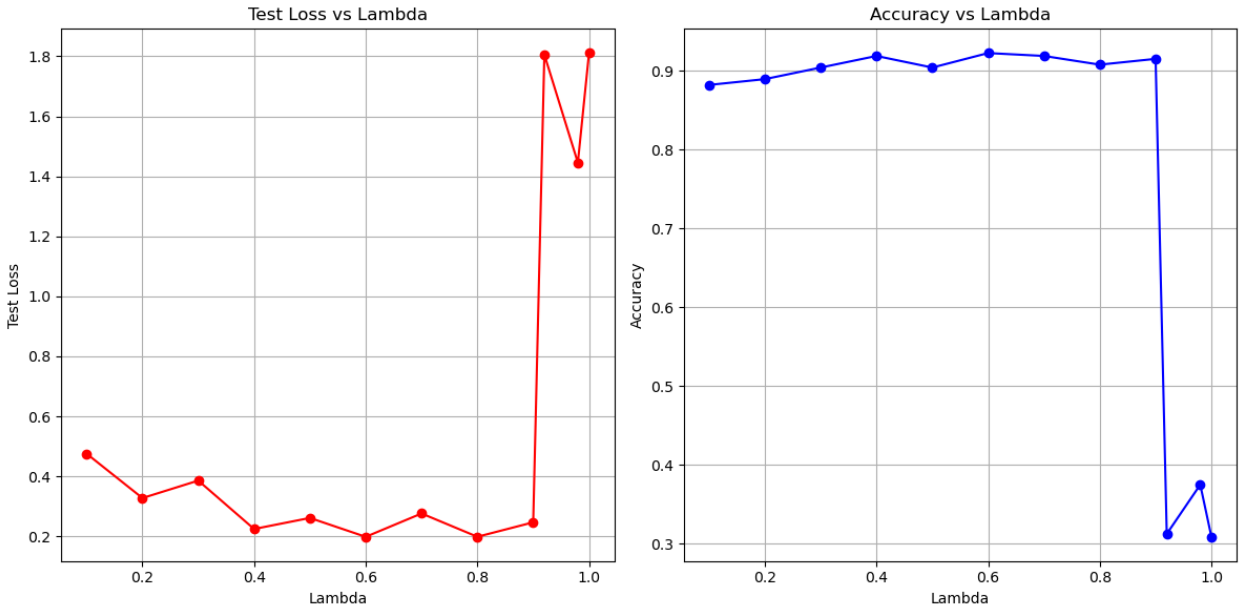

Figure 4.8: Impact of varying lambda on embeddings in the Cora dataset.

Table 4.4: Test Loss and Accuracy for Different Lambda Values on Cora Dataset

Lambda	**Test Loss**	**Accuracy**
0.1	0.4756	88.2353%
0.2	0.3278	88.9706%
0.3	0.3860	90.4412%
0.4	0.2247	91.9118%
0.5	0.2615	90.4412%
0.6	0.1986	92.2794%
0.7	0.2764	91.9118%
0.8	0.1988	90.8088%
0.9	0.2469	91.5441%
0.92	1.8055	31.2500%
0.98	1.4456	37.5000%
1.0	1.8109	30.8824%

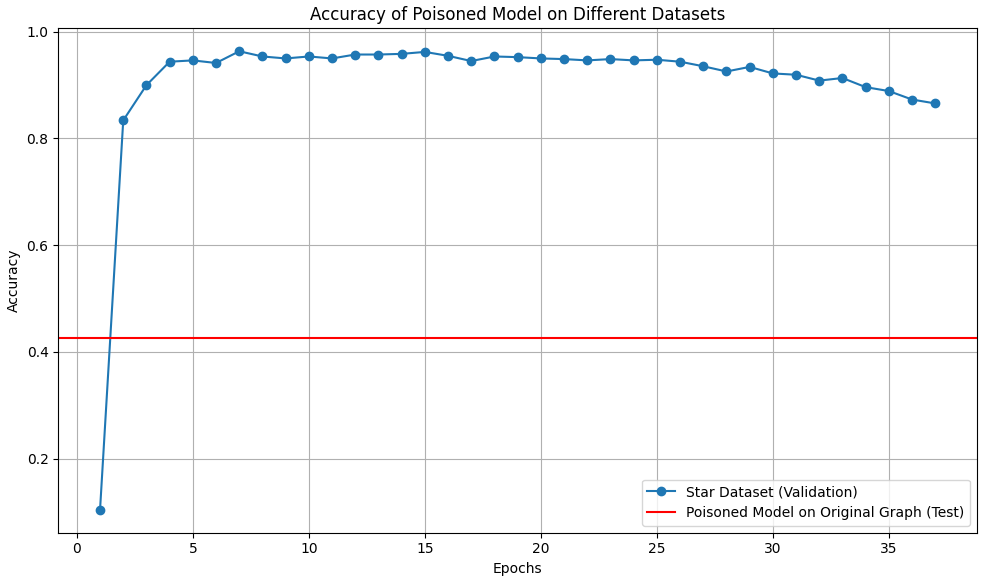

Figure 4.9: Performance comparison of the model on the original dataset before and after the poisoning attack.

Chapter 5

Conclusion & Future Work

5.1 Conclusion and Limitation

This study addressed an issue in the realm of graph datasets: executing attacks using node embeddings derived from textual data, a scenario that holds particular significance in social network analysis. Our experimental framework included both the text dataset: Hellaswag and graph datasets: Cora and CiteSeer. We comprehensively explored both decision-time and poisoning attacks, with a specific focus on the latter. Our approach diverged from standard methods, such as those in Meta Attack[25] and Net Attack[24], by exclusively altering node embeddings without affecting the graph structure and node attributes simultaneously. The results from our investigation demonstrate that, even within these constraints, it is possible to effectively undermine graph embeddings.

Notably, we observed that decision-time attacks exhibit a more pronounced impact compared to poisoning attacks. This could be attributed to the direct application of decision-time attacks on datasets, in contrast to poisoning attacks, which target models indirectly. Consequently, decision-time attacks can significantly alter results, highlighting their potency.

However, our findings also suggest that to achieve more substantial adversarial effects, stronger attack constraints might be necessary. This includes implementing higher values of ϵ or λ. We speculate that the resilience seen in our experiments may be partially due to the informational content inherent in the links of the graph, potentially offsetting the node embedding perturbations to some degree.

This research contributes to a deeper understanding of adversarial vulnerabilities in graph-based models, especially those using text-derived node embeddings, and lays the groundwork for further exploration in this important area of adversarial machine learning.

5.2 Future work

1. *Backdoor Attacks for Graph Data:* Future research could explore backdoor attacks on graphs. The aim would be to balance maintaining the performance on adversarial examples while preserving the original dataset's integrity and performance.

2. *Textual Embedding Revision:* Another intriguing avenue involves revising embeddings to enable backpropagation to the original text. This approach would assess if attacks adversely impact the text dataset. The challenge lies in converting tensor data back into coherent text, akin to image restoration techniques used in models like Clip [12], but specifically tailored for textual content.

3. *Exploring Defensive Strategies:* Additionally, investigating defensive mechanisms against these attacks could shed light on the robustness of graph-based models. This includes developing techniques to detect and mitigate adversarial modifications without compromising the model's performance on genuine data.

This research lays the groundwork for further exploration into the vulnerabilities of graph-based models, especially those utilizing text-derived embeddings, and opens new frontiers for both offensive and defensive strategies in adversarial machine learning.

References

[1] T. Balaji, C. S. R. Annavarapu, and A. Bablani. Machine learning algorithms for social media analysis: A survey. *Computer Science Review*, 40:100395, 2021.

[2] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Pattern Recognition*. Elsevier, 2018.

[3] C. M. Bishop. Pattern recognition and machine learning. 2006.

[4] 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.

[5] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. PMLR, 2018.

[6] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[7] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[8] L. Huang, D. Ma, S. Li, X. Zhang, and H. Wang. Text level graph neural network for text classification. *arXiv preprint arXiv:1910.02356*, 2019.

[9] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[10] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[11] I. Portugal, P. Alencar, and D. Cowan. The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications*, 97:205–227, 2018.

[12] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[13] Y. Sun, S. Wang, X. Tang, T.-Y. Hsieh, and V. Honavar. Adversarial attacks on graph neural networks via node injections: A hierarchical reinforcement learning approach. In *Proceedings of the Web Conference 2020*, pages 673–683, 2020.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[15] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2018.

[16] Y. Vorobeychik and M. Kantarcioglu. *Adversarial machine learning / Yevgeniy Vorobeychik, Murat Kantarcioglu.* Synthesis lectures on artificial intelligence and machine learning, 38. Morgan Claypool, San Rafael, California, 2018.

[17] Y. Wang, Y. Wang, J. Yang, and Z. Lin. Dissecting the diffusion process in linear graph convolutional networks. *CoRR*, abs/2102.10739, 2021.

[18] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu. Adversarial examples on graph data: Deep insights into attack and defense. *arXiv preprint arXiv:1903.01610*, 2019.

[19] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[20] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[21] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[22] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[23] Y. Zhu, X. Ai, Y. Vorobeychik, and K. Zhou. Homophily-driven sanitation view for robust graph contrastive learning, 2023.

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

[25] D. Zügner and S. Günnemann. Adversarial attacks on graph neural networks via meta learning. In *International Conference on Learning Representations*, 2019.