Universität des Saarlandes
MI Fakultät für Mathematik und Informatik
Department of Computer Science

Bachelorthesis

Link Stealing Attacks on Inductive Trained Graph Neural Networks

submitted by

Philipp Zimmermann

on January 01, 1970

Reviewers

Prof. Dr. Doktor Professer

Prof. Dr. Realy Intelligent

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Saarbrücken, January 01, 1970,

(Philipp Zimmermann)

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, January 01, 1970,

(Philipp Zimmermann)

Abstract

Since nowadays graphs are a common way to store and visualize data, Machine Learning algorithms have been improved to directly operate on them. In most cases the graph itself can be deemed confidential, since the owner of the data often spends much time and resources collecting and preparing the data. In our work, we show, that so called graph neural networks can reveal sensitive information about any graph they are queried on. We focused on extracting information about the edges of the target graph by observing the predictions of the target model in so called link stealing attacks. In prior work, He et al. proposed the first link stealing attacks on graph neural networks, focusing on the transductive learning. More precisely, given a black box access to a graph neural network, they were able to predict, whether two nodes of a graph that was used for training, are linked or not. We now focus on the inductive setting. Specifically, given a black box access to a graph neural network, we aim to predict whether there exists a link between any two nodes of any target graph, not only the one, the graph neural network was trained on. present results

Acknowledgements

Contents

Chapter 1

Introduction

1.1 Motivation

A graph is a datastructure which is used to model large data and the relationships between entities [1, 2]. It consists of nodes and edges and can be used to model data in almost every domain. For example in social networks, healthcare analytics or protein-protein interactions. In a social network, the nodes would be the users that are registered and the edges would represent whether the users know each other or not by connecting them or not. A graph itself can be deemed as intellectual property of the data owner, since she may spent lots of time and resources collecting and preparing the data. In most cases the graph is also highly confidential because it contains sensitive information like private social relationships between users in a social network or medical information about specific people in healthcare-analytic datasets. Since nowadays graphs are a common way to store and visualize data, Machine Learning algorithms have been improved to directly operate on them. These Machine Learning Models are called Graph Neural Networks (GNNs) [3, 4]. They can be used in different ways to operate on graphs. For example they can be trained to perform node classification [5]. More precisely, given a graph containing some labeled nodes the model is trained to predict the labels of the other unlabeled nodes in the graph. They can also be used to perform link prediction like in social networks where the friendship between two users is guessed [6].

A Graph Neural Network can be trained in different ways, depending on the purpose it will be used for. One way is to train them transductive [7–10]. Regarding the node classification problem that means, that test and evaluation node features are given during training. Only the labels are unknown. Nevertheless this training method is possible theoretically, it cannot be applied to real world problems like in social networks. That's why e.g. social networks keep evolving. Every day new users register and other user delete their accounts. For datasets like that GNNs can also be trained inductive [11–13]. Specifically, now not only the labels of the test and evaluation nodes is unknown but also their features and connections. That means, that the model is trained on one graph and will be evaluated on another one. In that way it is now possible to update the model on new nodes without retraining it over and over again on the full graph.

In our work, we show, that inductive trained Graph Neural Networks are very likely to leak sensitive information about any target graph it is queried on. Meaning that even if the target graph is unknown by the target model, it is still possible to extract sensitive link-information.

1.2 Outline

write at the end

Chapter 2

Related Work

Ever since machine learning algorithms were developed, there have been new attacks against these models. In 2004, Dalvi et al. proposed simple evasion attacks to defeat linear classifiers that are used in spam filters [14]. Later in 2006, Barreno et al. outline a broad taxonomy of attacks against linear classifier in their paper *Can Machine Learning Be Secure?*[15]. After in 2012 Deep Neural Networks began to dominate different domains, attacks against these models were also found and further developed [16, 17]. Today it is well know, that machine learning models are vulnerable in a security and privacy manner and that there exist many attacks against Machine Learning Models. With *Membership Inference Attacks* [18–22] an adversary aims to distinguish whether a given data sample was part of the training dataset of the target model or not. Shokri et al. [20] proposed the first Membership Inference Attack on Machine Learning Models. Given a data record and black-box access to a model, they were able to determine if the record was in the target models training dataset. The authors used adversarial machine learning to train an adversary model, that recognizes differences in the target models prediction. They evaluated their experiments on realistic datasets like a hospital discharge, whose membership is sensitive from the privacy perspective and showed that these models can be vulnerable to membership inference attacks. To prevent this attacks, many defenses have been proposed [20, 23–25]. With *Model Inversion Attacks* [26–29], an adversary aims to learn sensitive attributes of the target models training dataset. The first model inversion attack has been proposed by Fredrikson et al. [26]. They showed, that given the target model and some demographic information about a patient, it is possible to predict the patient's genetic markers. The authors further investigate, that differential privacy mechanisms prevent their model inversion attacks, when the privacy budget is carefully selected. With *Model Extraction Attacks* [30–32], an adversary aims to steal the model internals and uses this information to gradually train a substitute model that immitates the behaviour of the target. Tramèr et al. [32] proposed simple model extraction attacks, which were able to steal target models with near-perfect fidality. A similar aproach was proposed by Wang and Gong [33], who were able to successfully steal the hyperparameters of target models. To mitigate these attacks, many defenses have been proposed [31, 34–36]. For Example Juuti et al. [31], showed that they were able to detect all prior model extraction attacks with no false positives by raising an alarm when the distribution of consecutive API queries deviates from benign behavior. Hu and Pang [34] proposed an effective defense against model extraction attacks on Generative Adversarial Networks [37], considering a trade-off between the utility and security of GANs.

Since many real world problems can be represented as graphs, it was urgent to develop machine learning algorithms to fully utilize graph data. Therefor, so called Graph Neural Networks have been developed and already used in various tasks [3, 5, 38, 39]. Although, recent work shows, that graph neural networks are vulnerable to adversarial attacks as well [40–42, 42, 43]. More precisely, an adversary can decrease the targets accuracy by manipulating the graph structure or node features. For example, Sun et al. [41] proposed node injection poisoning attacks, where adversarial nodes are injected into existing graphs to reduce the performance of classifying existing nodes. Zügner et al. [42] showed that even with only a few pertubations the accuracy of node classification significantly drops, while focusing on training and testing phase. Wang et al. [40] focused on adversarial collective classification. They formulate their attack as a graph-based optimization problem, solving which produces the edges that an attacker needs to manipulate to achieve its attack goal and als propose several techniques to solve the optimization problem. Lastly Jin et al. [43] categorized existing attacks and defenses, and reviewed the corresponding state-of-the-art methods. They also have developed a repository with representative algorithms. Our work is different, since we focus on stealing links from graph neural networks.

In recent work, He et al. proposed the first attacks on Graph Neural Networks to obtain information about the underlaying graph [44]. They call their attacks *Link Stealing Attacks*. Given a black box access to a graph neural network, they showed that an adversary is able to predict whether any two nodes of a graph, that was used for training, are linked or not. The attacks reveal serious concerns on the intellectual property, confidentiality and privacy of graphs, when they are used for training. Our work is different, since we focus on *Link Stealing Attacks* on inductive trained Graph Neural Networks. Specifically, given a black box access to a graph neural network, we aim to predict whether there exists a link between any two nodes of any graph, not only the one, the graph neural network was trained on.

Chapter 3

Background

3.1 Neural Networks

Neural Networks (NNs) are key components in Artificial Intelligence (AI) and Deep Learning. They try to simulate some properties and the functionality of biological neural networks, like our brain, by imitating the way biological neural systems process data. Today, they have been applied successfully to speech recognition, face recognition on images or the transformation from speech to text. They are used to model software agents in video games, let autonomous robots learn new things or find patterns in data.

A neural network consists of multiple layers. An input layer, one or many more hidden layers and an output layer. Each of these layers contain multiple neurons, which are represented as mathematical functions, that take multiple inputs and use them to calculate one output. Such neuron is also called perceptron, while fully connected neural networks are called multilayer perceptron (MLP).

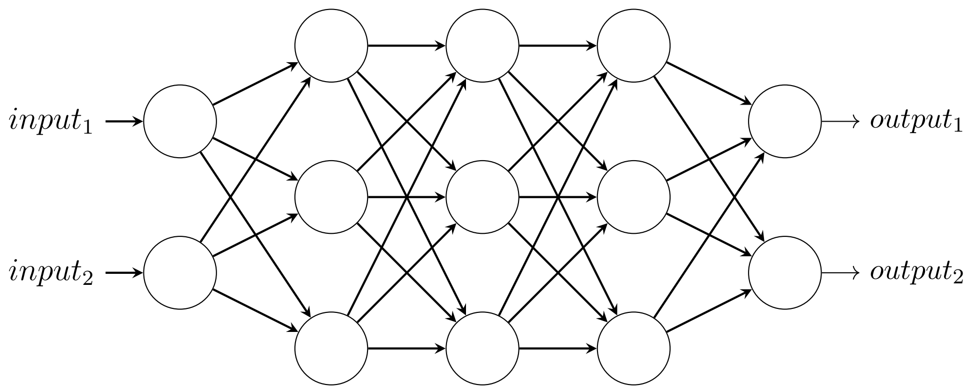

Figure 3.1: Multilayer Perceptron

Train a Neural Network

When we are confronted to a new task, we try to gain as much information as possible and based on them, we learn how to solve the problem. Neural networks behave similar. Before we can apply a neural network on classifying whether the object we provide is a spoon or fork, we need to tell the network, based on which information it should make its decision. That could be images of spoons and forks or their weight, length and width. We call this data *Training Data*. To train the model in our example, we provide an image of a spoon and let the network make its decision. If the decision is correct, we wont change anything. But if the decision is incorrect, we need to slightly modify the weights - the connections between the perceptrons - to let the model behave different in the future. We do this for each image in our training set and repeat this process n times (for n epochs). After the training, we hopefully have a good trained neural network, which performs well on the provided task.

3.2 Graphs

As Graph we denote a data structure that contains nodes and edges. Let $G = (V, E)$ be a graph with V being the set of nodes and E being the set of edges. We denote $\vec{a}$ as the feature vector of node a, representing the attributes of a. An edge $e = (i, j)$ contains the source node i and the destination node j. In that way, links describe the relationship between the source and destination node. The most popular example where graphs are used to model data, are social networks. The nodes represent the users that have multiple attributes like location, gender, work place etc., while the edges state which relationship the users have. Lets consider a directed graph $G = (V, E)$ with V representing the users and E their relationships. If user a follows user b, the edge $e_{ab} = (a, b)$ would be an element in E. If user b follows user a, the edge $e_{ba} = (b, a)$ would be an element in E. Lets consider an undirected graph $G' = (V', E')$ with V' representing the users and E' their relationships. In G' it doesn't matter whether user a' follows b' or the other way around. Both results in $e'_{a'b'}, e'_{b'a'} \in E'$. Figure 3.2 shows an undirected graph. Since there is a link drawn between node A and node B, we know, that there exists some relationship between them, but we don't know who follows whom $(e_{AB}, e_{BA} \in E)$. The same holds for $e_{CF}, e_{FC} \in E$ or $e_{EG}, e_{GE} \in E$. Since there isn't a link drawn between node E and node C, there doesn't exists a known relationship between them $(e_{EC}, e_{CE} \notin E)$.

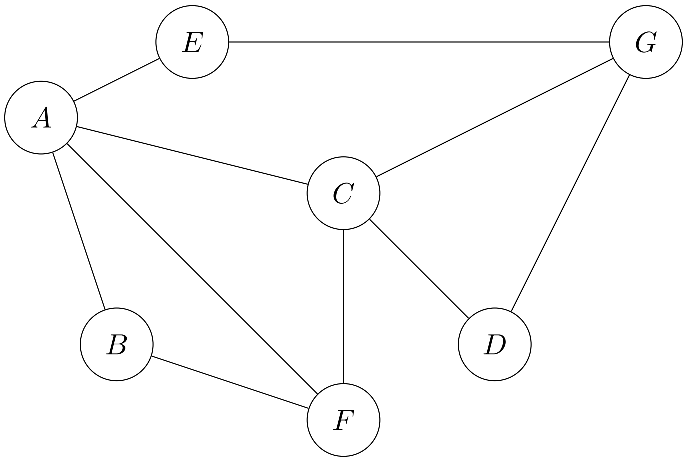

Figure 3.2: Undirected Graph

3.3 Graph Neural Networks

In the last decades, social networks have become a huge part of life for many people all around the world. The companies behind these networks collect tons of data of each user every day. Let $G = (V, E)$ be a graph that models such a social network. The set of all users is given as V and E represents their relationships. The feature vector $\vec{a}$ of user a contains all information the company already has regarding this user. That could be name, relationship status, workplace, address, amount of children and so on. Lets consider one attribute (*workplace*) as non mandatory for the registration. This leads to some users $v_{workplace} = \{v \mid \forall v \in V : v \text{ has } workplace \text{ given}\}$, the company knows the workplace from and some users $v_{unknown} = \{v \mid \forall v \in V : v \text{ has } workplace \text{ not given}\}$, where the attribute is unknown. Based on the information $v_{workplace}$ provides, a graph neural network model f can be trained to predict the missing attribute of all users in $v_{unknown}$. This is a simple node classification task, where the label is the missing attribute *worplace*. There exist two major ideas of training a graph neural network - transductive or inductive. In our experiments introduced and described in Chapter 4 we focus on attacks on inductive trained graph neural networks.

Train a Graph Neural Network

To describe the process of training a graph neural network, we will stick to the example above. Let f be the GNN we want to train on an undirected graph $G = (V, E)$ to perform some node classification task. We denote *workplace* as label, that we want to predict and $V = v_{workplace} \cup v_{unknown}$ as set of nodes, containing every node, no matter whether the label is given or not. As our training dataset we define $d_{train} = \{(v, v_n) \mid \forall v \in v_{workplace} : v_n = neighborhood(v)\}$, where $neighborhood(n)$ is a function defined to collect the k-hop neighborhood of n and aggregates their feature vectors. $k = 0$ only considers n and no neighbors of n. $k = 1$ also includes the neighbors of n, while $k = 2$ also aggregates the neighbors of the neighbors of n. Depending on the type of GNN, this aggregation function $neighborhood()$ can differ in the algorithm collecting the neighborhood vectors.

Transductive Learning

For transductive learning we consider all information as given, especially while training the graph neural network. More precisely, we denote, as mentioned before, $V = v_{workplace} \cup v_{unknown}$. This means, that for training the model on d_{train}, which contains all labeled nodes, the feature vectors of nodes, that are not labeled are included as well. Later when the model will be evaluated and tested, it already has seen the feature vectors of the nodes it should predict the labels of. So, the information of the nodes, the model needs to classify, is already known during the training phase.

Inductive Learning

For inductive learning, we only consider the information of nodes that are labeled given while training the graph neural network. More precisely, we denote $V = v_{workplace}$ and $d_{train} = \{(v, v_n) \mid \forall v \in V : v_n = neighborhood(v)\}$. This means, that for training the model on d_{train} only labeled nodes are used. Lets assume that three new users register to the social network, nobody setting his / her workplace. Since some other mandatory information and maybe some links are given, the graph neural network can be used to predict the workplace of the new users nevertheless they haven't been included in the training process. So, it is possible to train the graph neural network on some graph and apply it on others, since it is easy adjustable.

Since the most real world problems like friendship prediction in social networks or protein-protein interactions in chemical networks cannot be modeled with a static graph, the inductive learning method is commonly used, while the transductive one isn't. In our work, we show that given an inductive trained graph neural network, that was trained on a graph G_1, we are able to extract information of another graph G_2 that is completely unknown to the model. More precisely, we steal links from G_2 using the prediction posteriors of the model while querying it on G_2.

Chapter 4

Attacks

In recent work He et al. [44] proposed the first link stealing attacks on graph neural networks. They focused on stealing links of the graph, that was used for training the given target model. Like described in Section 3.3.1 this is an attack on transductive trained graph neural networks. In our work, we want to show, that it is possible for an adversary to steal links from any graphs, given black-box access to an inductive trained target graph neural network model.

4.1 Adversary's Goal

Let f_t be the target graph neural network model, trained to perform some machine learning task. Let $G_s = (V_s, E_s)$ be a graph with $|V_s|$ nodes and $|E_s|$ edges. We assume, that some of G_s's links/edges are missing. The goal of an adversary A is, to infer whether two nodes $i, j \in V_s$ are connected to each other or not. More precisely, whether the link (i, j) between the nodes i and j is missing, or does not exist.

4.2 Intuition

Since A has black box access to its target model, it will use the posterior output of f_t, to make its classification. To do so, A queries f_t on two nodes $i, j \in V_s$, from which it want's to know whether they are linked or not. For both nodes f_t will return a posterior: $post_i = f_t(G_s, i)$ and $post_j = f_t(G_s, j)$. A can be trained on how the two posteriors "look like", when i and j originally have been connected and the edge is missing in G_s or how they look like when there is no edge to recover.

4.3 Threat Model

For any of our attacks, we assume, *Black-Box Access* (Query Access) to the target graph neural network model f_t, that was trained on a graph dataset D_{f_t}. We consider f_A another graph neural network model, which was trained by the adversary using a graph dataset D_{f_A}. The adversary A was trained on a dataset D_A to perform link stealing attacks. We denote $G_A = (V_A, E_A)$ as graph used by the adversary querying f_A to sample D_A. We assume, that in any attack D_{f_t} and G_s are from the same dataset distribution, as well as D_{f_A} and G_A share the same one. However, D_{f_A} must not be from the same dataset distribution as D_{f_t}. For *Attack 1* and *Attack 2* we consider $f_t = f_A$, which implies $D_{f_t} = D_{f_A}$. Table 8.2 can be used to look up notation descriptions.

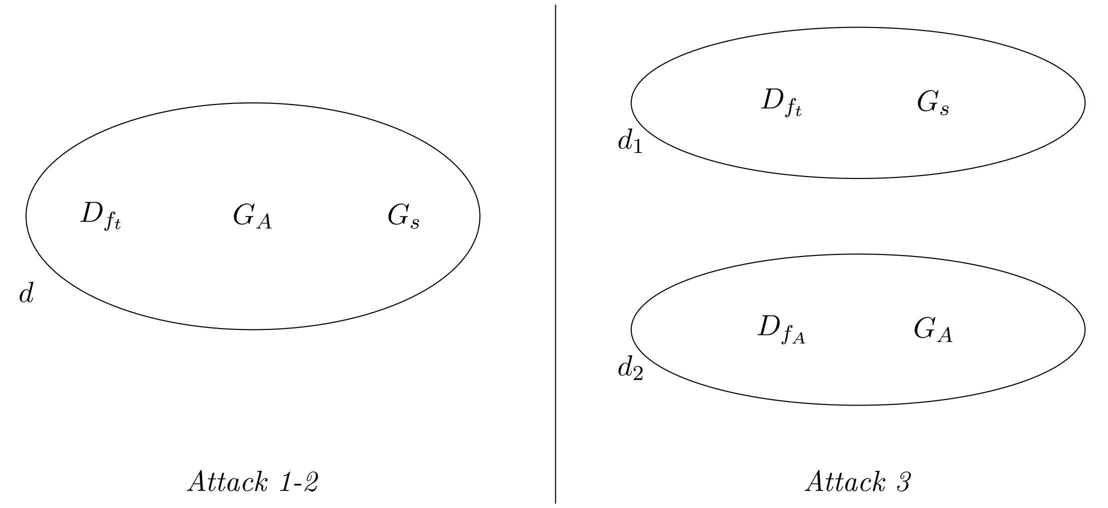

Figure 4.1: Dataset Distributions for Link Stealing Attacks

4.4 Attack Methodology

Let f_t be the target graph neural network model and $G_s = (V_s, E_s)$ a graph with $|V_s|$ nodes and $|E_s|$ edges. We assume that E_s is not complete. More precisely, there exists an edge (i, j) between the nodes $i, j \in V_s$, but $(i, j) \notin E_s$. The adversary A queries f_t on both nodes i and j, obtaining $post_i = f_t(G_s, i)$ and $post_j = f_t(G_s, j)$.

4.4.1 Attack 1

In *Attack 1* we consider G_A and D_{f_t} from the same dataset distribution and denote $f_t = f_A$. Meaning, that A samples its training dataset D_A by querying f_t. That can be done, because $|post_u| = |f_t(G_A, u)|$ (training phase) and $|post_v| = |f_t(G_s, v)|$

(attack phase), with $u \in V_A$ and $v \in V_s$, have the same dimension. Based on this assumption, A can directly be trained on the posteriors generated by f_t. Therefore we concatenate $post_i$ and $post_j$ obtaining the input $post_{ij} = cat(post_i, post_j)$, with $cat(A, B) = [a_0, ..., a_n, b_0, ..., b_n]$, where $A = [a_0, ..., a_n]$ and $B = [b_0, ..., b_n]$. Given $post_{ij}$, A can infer, whether i and j have been connected and the edge is missing in G_s or not.

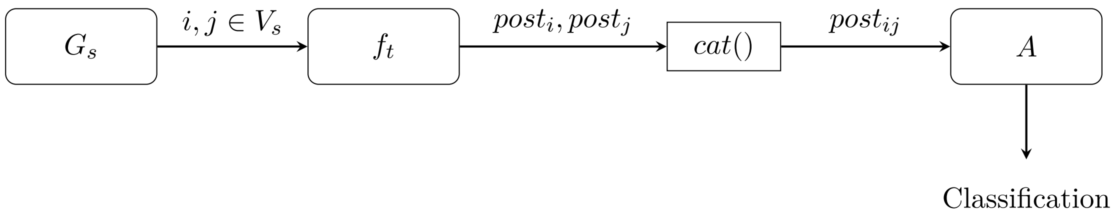

Figure 4.2: Flow Chart - Attack 1

4.4.2 Attack 3

In *Attack 3* we consider G_A and D_{f_t} from different dataset distributions. So the adversary trains another graph neural network model f_A with a dataset D_{f_A}. Since f_t and f_A are trained on different dataset distributions, we must assume that they have different parameters like feature amount or number of classes. Meaning, that it is not possible anymore, to train the adversary directly on the posterior output of f_t, since $|post_u| = |f_A(G_A, u)|$ (training phase) and $|post_v| = |f_t(G_s, v)|$ (attack phase), with $u \in V_A$ and $v \in V_s$, have different dimensions. Based on this assumption, we need to sample the input for A, by creating features based on the posteriors, instead of using them directly. As features we use eight common distance metrics, to measure the distance between $post_i$ and $post_j$. We have in total experimented with Cosine distance, Euclidean distance, Correlation distance, Chebyshev distance, Braycurtis distance, Canberra distance, Manhattan distance, and Square-euclidean distance. The formal definition of each distance metrics is listed in table 8.1. So we construct the input $dist_{ij}$ for A as follows: $dist_{ij} = dist(post_i, post_j)$, where $dist(post_i, post_j) = [Cosine(post_i, post_j), ..., Sqeuclidean(post_i, post_j)]$. Given $dist_{ij}$, A now can infer, whether i and j have been connected and the edge is missing in G_s or not.

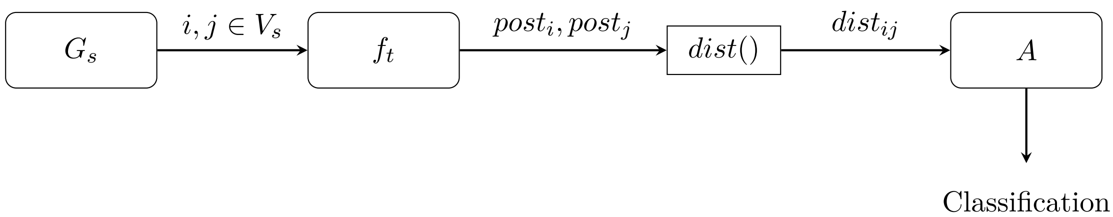

Figure 4.3: Flow Chart - Attack 3

4.4.3 Attack 2

In *Attack 2* we consider G_A and D_{f_t} from the same dataset distribution and denote $f_t = f_A$ like it was done in *Attack 1*. However, for better comparison of the impact of the dataset distribution, we sample the input for A, like it was done in *Attack 3*.

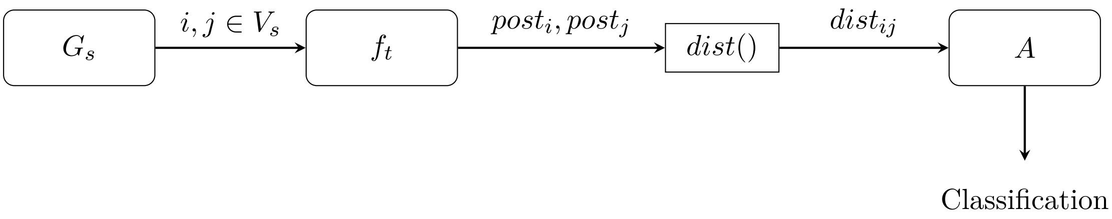

Figure 4.4: Flow Chart - Attack 2

Furthermore, for each Attack we assume different amounts of edges given in G_A. The percentage of known edges is denoted as α and has an impact on the prediction (posteriors) of f_t and f_A. Therefor we construct new adversary graphs: $G_A^\alpha = (V_A^\alpha, E_A^\alpha)$ with $|E_A^\alpha| = \alpha * |E_A|$ and $V_A^\alpha = V_A$. As different stages we define $\alpha = 0.0, 0.2, 0.4, 0.6, 0.8$. The first case, $\alpha = 0.0$ represents an adversary graph $G_A^{0.0} = (V_A^{0.0}, E_A^{0.0})$ without any edges / no knowledge of the relationship between the nodes. $\alpha = 0.8$ leads to an adversary graph $G_A^{0.8} = (V_A^{0.8}, E_A^{0.8})$ with almost every edge considered to be known.

The following table shows all three attacks with the dataset distribution, the input for the adversary A and the feature amount of A.

Attacks	Dataset Distribution	Input for A	A's Feature Amount
Attack 1	Same	$inp_{ij} = cat(post_i, post_j)$	$\lvert inp_{ij}\rvert = \lvert post_i\rvert + \lvert post_j\rvert = 2 * \lvert post_i\rvert$
Attack 2	Same	$inp_{ij} = dist(post_i, post_j)$	$\lvert inp_{ij}\rvert = 8$
Attack 3	Different	$inp_{ij} = dist(post_i, post_j)$	$\lvert inp_{ij}\rvert = 8$

Table 4.1: Attack Methodology

Chapter 5

Implementation

In order to analyze how effective our attacks can steal links from graphs that are used to query inductive trained graph neural networks, we performed several experiments. In this chapter we want to go through their implementation. We covered multiple datasets and types of graph neural network models, leading to an amount of ~200 experiments. Due computational and time constraints, most of the parameters to optimize the attacks remain unexplored.

5.1 Datasets

For all our experiments, we used 3 datasets in total. In the table below, they are listed with their most interesting attributes.

Name	Number of Nodes	Number of Edges	Number of Classes	Feature Amount
Cora	2708	5429	7	1433
CiteSeer	3327	4732	6	3703
Pubmed	19717	44338	3	500

Table 5.1: Dataset Information

5.1.1 Sample Datasets for Experiments

In order to fulfill the criteria that G_s and D_{f_t} are from the same dataset distribution like described in Section 4.3, we split our datasets and sampled new ones based on our original three datasets *Cora, CiteSeer* and *Pubmed.*

Same Dataset Distributions - Attack 1-2

Let D be one of our three original datasets. We can obtain two subgraphs by splitting D into D_{f_t} and D'. D_{f_t} is used to train our target model f_t, while D' is used to sample G_A, which is a graph, that was modified by deleting some known edges to simulate an incomplete graph. D_A is obtained by querying f_t on a subgraph of G_A.

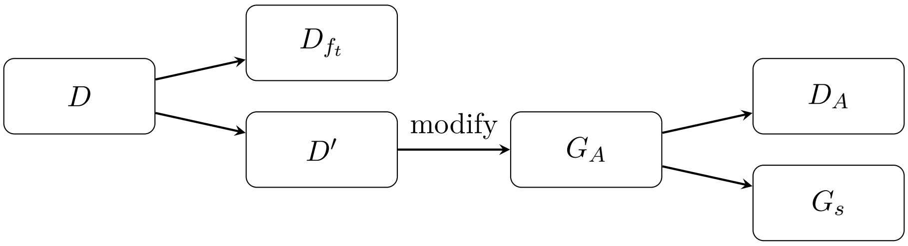

Figure 5.1: Sampling Datasets - Same Dataset Distribution - Attack 1-2

Different Dataset Distributions - Attack 3

Let D_1 and D_2 be two of our three original datasets. D_1 is used to sample D_{f_A} and G_A. D_{f_A} is used to train the adversary GNN f_A, while G_A is used to query f_A to obtain D_A, which will be used to train our attacker model A. D_2 is used to sample D_{f_t} and G_s. D_{f_t} is used to train our target model f_t, while G_s is the incomplete graph, the adversary wants to steal links from.

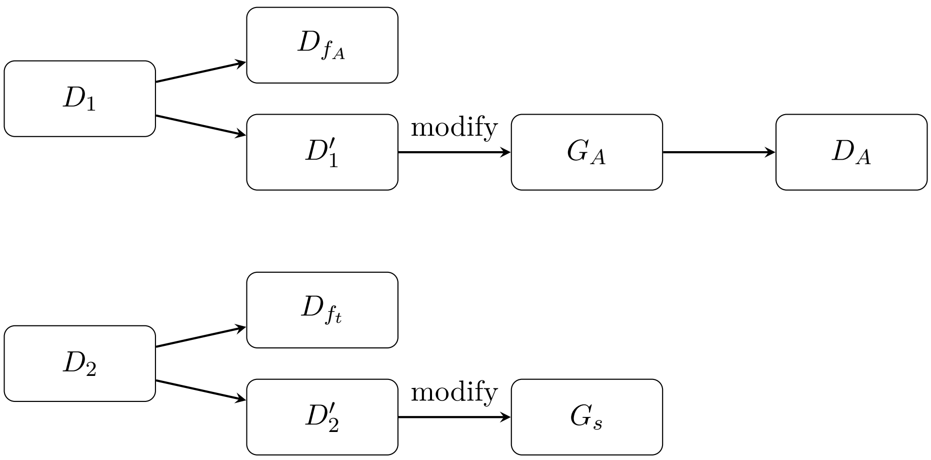

Figure 5.2: Sampling Datasets - Different Dataset Distribution - Attack 3

5.2 Target Models

As our target models, we used three different types of graph neural network models, each with a slightly different algorithm used to obtain the neighborhood embeddings.

5.2.1 GraphSAGE

In June 2017 Hamilton et al.[45] proposed a general framework, called GraphSAGE (SAmple and aggreGatE), for inductive node embedding. They came up with an idea of leveraging node features like text attributes, node profile information or node degrees to learn an embedding function that generalizes to unseen nodes instead of prior approaches that use matrix factorization. Until then, the training process focused on individual embeddings for each node, but with the GraphSAGE algorithm, a function is learned that generates embeddings by sampling and aggregating features from a node's neighborhood.

Each of our GraphSAGE-Target Models has been trained on one of our datasets

5.2.2 Graph Attention Networks

pass

5.2.3 Graph Convolutional Networks

pass

For each

5.3 Attacker Model

In recent work He et al. [44] proposed the first link stealing attacks on graph neural networks. They focused on stealing links of the graph, that was used for training the given target model. Like described in Section 3.3.1 this is an attack on transductive trained graph neural networks. In our work, we want to show, that it is possible for an adversary to steal links from any graphs, given black-box access to an inductive trained target graph neural network model.

Adversary's Goal

Let f be the target graph neural network model and $G_{adv} = (V_{adv}, E_{adv})$ a graph with $|V_{adv}|$ nodes and $|E_{adv}|$ edges. We assume, that some of G_{adv}'s links/edges are missing. The goal of an adversary is, to infer whether two nodes i and j are connected to each other or not. More precisely, whether the link (i, j) between node i and node j is missing or does not exist.

5.4 Attack 1

In this section, we propose our first attack. Given a target graph neural network model f and a graph G_{adv} of the same dataset distribution as f's training graph, an adversary a aims to steal missing edges of G_{adv}. Therefor it uses the posterior output $f(i)$ and $f(j)$ of two nodes i and j and concatenates them to get the input feature $post_{ij} = concat(f(i), f(j))$. The adversary a is then trained on $post_{ij}$.

5.4.1 Thread Model

In this attack the adversary a has *Black-Box Access* (Query-Access) to the target model f. The graph G_{adv} is from the same dataset distribution. However, f wasn't trained on G_{adv}.

5.4.2 Attack Methodology

To perform this attack, we split a given dataset $G = (V, E)$, into one traingraph $G_{train} = (V_{train}, E_{train})$ and one testgraph $G_{test} = (V_{test}, E_{test})$. We sample the traingraph as $V_{train} = \{i | \forall i \in V : random(0, 1) == 1\}$, where $random(0, 1)$ returns the values 0 or 1 at random, leading to a random split of the nodes. $E_{train} = \{(i, j) | \forall (i, j) \in E : i, j \in V_{train}\}$ now contains the edge (i, j) if both nodes i and j are in V_{train}. The testgraph is now sampled similary. $V_{test} = \{j | \forall j \in V : j \notin V_{train}\}$ and $E_{test} = \{(i, j) | \forall (i, j) \in E : i, j \in V_{test}\}$

5.4.2.1 Target Model

The target model f is now trained on G_{train} to perform a node classification task. Especially, given a node's features, its neighbors' and the edges between them, f outputs a prediction posterior of the class.

5.4.2.2 Attacker Model

We first create a raw dataset d_{raw} based on G_{test}. To do so, we create a clone of the testgraph $G_{adv} = G_{test}$, which will represent the adversary's graph. We now collect a set of positive samples $pos = \{(i, j, 1)|\forall i, j \in V_{test} : (i, j) \in E_{test} \wedge |pos| < ((1-\alpha) * |E_{test}|))\}$, containing pairs of nodes, that are connected in the testgraph, where α denotes the percentage of known edges. We then delete all edges we sampled, in our graph clone $E_{adv} = \{(i, j)|\forall(i, j) \in E_{adv} : (i, j) \notin pos\}$, to represent the missing edges, we want to steal. Now, we collect a set of negative samples $neg = \{(i, j, 0)|\forall i, j \in V_{test} : (i, j) \notin E_{test} \wedge |neg| < ((1-\alpha) * |E_{test}|))\}$, containing pairs of nodes, that are not connected in G_{test}. Our raw dataset $da\text{-}raw = pos \cup neg$, now contains positive and negative samples obtained from G_{test}. As the next step, we create the adversary's dataset $da = \{(post_{ij}, l)|\forall(i, j, l) \in da\text{-}raw : post_{ij} = concat(target(G_{adv}, i), target(G_{adv}, j))\}$. $target(G_adv, i)$ returns the node classification output posterior of the target model, when it is queried on i given the adversary's graph G_{adv}. $concat(a, b)$ concatenates the output posteriors a and b with each other returning the feature we will train the attacker model on. l denotes the label either being 1 (positive sample) or 0 (negative sample). With our adversary's dataset da we can now continue training our attacker model using $post_{ij}$ as input features and l as class.

5.5 Attack 2

In this section, we propose our second attack. Given a target graph neural network and a graph of the same dataset, that wasn't used for training the target model, an adversary aims to steal missing edges of its graph. Therefor it uses the posterior output of the two nodes, it queries the network on and calculates the distance between these two vectors in eight different ways and uses these values as input features for training the attacker model.

5.5.1 Thread Model

The Thread Model for this attack is the same one described in Section 4.1.1.

5.5.2 Attack Methodology

Most of the Attack Methodology is the same as the one described in Section 4.1.2. There is one difference however. Instead of using the concatenation of the two output posteriors, we now use them as vectors, to calculate their distances in eight different ways. We have in total experimented with 8 common distance metrics: Cosine distance, Euclidean distance, Correlation distance, Chebyshev distance, Braycurtis distance, Canberra distance, Manhattan distance, and Square-euclidean distance.

5.5.2.1 Attacker

We first create *da-raw* like described in Section 4.1.2.2. Our adversary's dataset can now be desribed as the following. $da = \{(dist_{ij}, l) | \forall (i, j, l) \in \textit{da-raw} : dist_{ij} = d(target(G_{adv}, i), target(G_{adv}, j))\}$, where $d(a, b) = concat(dist_1(a, b), ..., dist_8(a, b))$ and l again denotes the label. With our adversary's dataset da we can now continue training our attacker model using $dist_{ij}$ as input features and l as class.

5.6 Attack 3

In this section, we propose our last attack. Given a target graph neural network and a graph of a different dataset, that wasn't used for training the target model, an adversary aims to steal missing edges of its graph. Therefor it uses the posterior output of the two nodes, it queries the network on and calculates the distance between these two vectors in eight different ways and uses these values as input features for training the attacker model.

5.6.1 Thread Model

In this attack the adversary has *Black-Box Access* (Query-Access) to the target model and uses a different source dataset than the target.

5.6.2 Attack Methodology

As mentioned before, we now have two different datasets $G_{target} = (V_{target}, E_{target})$ and $G_{attacker\_model} = (V_{attacker\_model}, E_{attacker\_model})$.

5.6.2.1 Target

The target model is now trained on G_{target} to perform node classification. Especially,given a node's features, its neighbors' and the edges between them, the model outputs aprediction posterior of the class.

5.6.2.2 Attacker Model

We first create the raw dataset *da-raw* the same way, we did before but this time with $G_{attacker\_model}$. To do so, we again create a clone $G_{adv} = G_{attacker\_model}$. We now collect a set of positive samples $pos = \{(i, j, 1) | \forall i, j \in V_{attacker\_model} : (i, j) \in E_{attacker\_model} \wedge |pos| < ((1 - \alpha) * |E_{attacker\_model}|))\}$. We then delete all edges we sampled, in our graph clone $E_{attacker\_model} = \{(i, j) | \forall (i, j) \in E_{adv} : (i, j) \notin pos\}$, to represent the missing edges, we want to steal. Now, we collect a set of negative samples $neg = \{(i, j, 0) | \forall i, j \in V_{attacker\_model} : (i, j) \notin E_{test} \wedge |neg| < ((1 - \alpha) * |E_{attacker\_model}|))\}$, containing pairs of nodes, that are not connected in $G_{attacker\_model}$. Our raw dataset $da\text{-}raw = pos \cup neg$, now contains positive and negative samples obtained from $G_{attacker\_model}$. As the next step, we create the adversary's dataset $da = \{(dist_{ij}, l) | \forall (i, j, l) \in da\text{-}raw : dist_{ij} = d(target(G_{adv}, i), target(G_{adv}, j))\}$, where $d(a, b) = concat(dist_1(a, b), ..., dist_8(a, b))$ and l again denotes the label. With our adversary's dataset da we can now continue training our attacker model using $dist_{ij}$ as input features and l as class.

Chapter 6

Evaluation

Chapter 7

Discussion

Chapter 8

Conclusion

List of Figures

List of Tables

Appendix

Update table

Metrics	Definition
Cosine	$1 - \frac{f_t(u) \cdot f_t(v)}{\lvert f_t(u)\rvert_2 \lvert f_t(v)\rvert_2}$
Euclidean	$\lvert f_t(u) - f_t(v)\rvert_2$
Correlation	$1 - \frac{(f_t(u) - \overline{f_t(u)}) \cdot (f_t(v) - \overline{f_t(v)})}{\lvert(f_t(u) - \overline{f_t(u)})\rvert_2 \lvert(f_t(v) - \overline{f_t(v)})\rvert_2}$
Chebyshev	$\max_i \lvert f_{t_i}(u) - f_{t_i}(v)\rvert$
Braycurtis	$\frac{\sum \lvert f_{t_i}(u) - f_{t_i}(v)\rvert}{\sum \lvert f_{t_i}(u) + f_{t_i}(v)\rvert}$
Manhattan	$\sum i \lvert f_{t_i}(u) - f_{t_i}(v)\rvert$
Canberra	$\sum_i \frac{\lvert f_{t_i}(u) - f_{t_i}(v)\rvert}{\lvert f_{t_i}(u)\rvert + \lvert f_{t_i}(v)\rvert}$
Sqeuclidean	$\lvert f_t(u) - f_t(v)\rvert_2^2$

Table 8.1: Distance metrics: $f_{t_i}(u)$ represents the i-th component of $f_t(u)$.

Notation	Description
f_t	Target GNN Model
D_{f_t}	Dataset used to train f_t
A	Adversary
D_A	Dataset used to train A
G_A	Graph used to sample D_A
α	Percentage of known edges in G_A
$G_A^{0.4}$	Graph used to sample D_A with 40% known edges
f_A	GNN Model, that is involved in training A
D_{f_A}	Dataset used to train f_A
G_s	Incomplete Graph, A performs link stealing attacks on

Table 8.2: Notations

Bibliography

[1] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *CoRR*, vol. abs/2005.00687, 2020. [Online]. Available: https://arxiv.org/abs/2005.00687

[2] D. J. Cook and L. B. Holder, *Mining graph data.* John Wiley & Sons, 2006.

[3] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," 2016.

[4] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," 2017.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[6] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," 2018.

[7] W. Liu and S.-F. Chang, "Robust multi-class transductive learning with graphs," pp. 381–388, 2009.

[8] Y. Zha, Y. Yang, and D. Bi, "Graph-based transductive learning for robust visual tracking," *Pattern Recognition*, vol. 43, no. 1, pp. 187–196, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320309002581

[9] Z. Wang, X. Zhu, E. Adeli, Y. Zhu, F. Nie, B. Munsell, and G. Wu, "Multi-modal classification of neurodegenerative disease by progressive graph-based transductive learning," *Medical Image Analysis*, vol. 39, pp. 218–230, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361841517300749

[10] P. P. Talukdar and K. Crammer, "New regularized algorithms for transductive learning," in *Machine Learning and Knowledge Discovery in Databases*, W. Buntine, M. Grobelnik, D. Mladenić, and J. Shawe-Taylor, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 442–457.

[11] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," 2020.

[12] R. A. Rossi, R. Zhou, and N. K. Ahmed, "Deep inductive graph representation learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 3, pp. 438–452, 2020.

[13] Y. Zhang, X. Yu, Z. Cui, S. Wu, Z. Wen, and L. Wang, "Every document owns its structure: Inductive text classification via graph neural networks," 2020.

[14] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial classification," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 99–108. [Online]. Available: https://doi.org/10.1145/1014052.1014066

[15] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 16–25. [Online]. Available: https://doi.org/10.1145/1128817.1128824

[16] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2014.

[17] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, p. 317–331, Dec 2018. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2018.07.023

[18] N. Carlini, C. Liu, Úlfar Erlingsson, J. Kos, and D. Song, "The secret sharer: Evaluating and testing unintended memorization in neural networks," 2019.

[19] Q. Chen, C. Xiang, M. Xue, B. Li, N. Borisov, D. Kaarfar, and H. Zhu, "Differentially private data generative models," 2018.

[20] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," pp. 3–18, 2017.

[21] S. Truex, L. Liu, M. E. Gursoy, L. Yu, and W. Wei, "Towards demystifying membership inference attacks," 2019.

[22] J. Hayes, L. Melis, G. Danezis, and E. D. Cristofaro, "Logan: Membership inference attacks against generative models," 2018.

[23] J. Jia, A. Salem, M. Backes, Y. Zhang, and N. Z. Gong, "Memguard: Defending against black-box membership inference attacks via adversarial examples," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications*

Security, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 259–274. [Online]. Available: https://doi.org/10.1145/3319535.3363201

[24] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models," 2018.

[25] J. Li, N. Li, and B. Ribeiro, "Membership inference attacks and defenses in classification models," *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, Apr 2021. [Online]. Available: http://dx.doi.org/10.1145/3422337.3447836

[26] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," *Proceedings of the ... USENIX Security Symposium. UNIX Security Symposium*, vol. 2014, p. 17—32, August 2014. [Online]. Available: https://europepmc.org/articles/PMC4827719

[27] S. Hidano, T. Murakami, S. Katsumata, S. Kiyomoto, and G. Hanaoka, "Model inversion attacks for prediction systems: Without knowledge of non-sensitive attributes," pp. 115–11 509, 2017.

[28] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1322–1333. [Online]. Available: https://doi.org/10.1145/2810103.2813677

[29] S. Chen, R. Jia, and G.-J. Qi, "Improved techniques for model inversion attacks," 2020.

[30] B. G. Atli, S. Szyller, M. Juuti, S. Marchal, and N. Asokan, "Extraction of complex dnn models: Real threat or boogeyman?" 2020.

[31] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, "Prada: Protecting against dnn model stealing attacks," 2019.

[32] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," 2016.

[33] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," pp. 36–52, 2018.

[34] H. Hu and J. Pang, "Model extraction and defenses on generative adversarial networks," 2021.

[35] H. Jia, C. A. Choquette-Choo, V. Chandrasekaran, and N. Papernot, "Entangled watermarks as a defense against model extraction," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/jia

[36] Y. Mori, A. Nitanda, and A. Takeda, "Bodame: Bilevel optimization for defense against model extraction," 2021.

[37] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[38] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[39] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *International Conference on Learning Representations*, 2018, accepted as poster. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[40] B. Wang and N. Gong, "Attacking graph-based classification via manipulating the graph structure," 03 2019.

[41] 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," New York, NY, USA, p. 673–683, 2020. [Online]. Available: https://doi.org/10.1145/3366423.3380149

[42] 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 and Data Mining*, Jul 2018. [Online]. Available: http://dx.doi.org/10.1145/3219819.3220078

[43] W. Jin, Y. Li, H. Xu, Y. Wang, S. Ji, C. Aggarwal, and J. Tang, "Adversarial attacks and defenses on graphs: A review, a tool and empirical studies," 2020.

[44] X. He, J. Jia, M. Backes, N. Z. Gong, and Y. Zhang, "Stealing links from graph neural networks," *CoRR*, vol. abs/2005.02131, 2020. [Online]. Available: https://arxiv.org/abs/2005.02131

[45] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.