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 September 14, 2021

Reviewers

Dr. Yang Zhang

Dr. Cristian-Alexandru Staicu

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, September 14, 2021,

(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, September 14, 2021,

(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 inductive trained graph neural networks can reveal sensitive information about their training graph. We focus on extracting information about the edges of the target training 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 setting. More precisely, given black box access to a graph neural network, the authors were able to predict, whether two nodes of a graph that was used for training the model, are linked or not. Since transductive trained Graph Neural Networks can only handle fixed graphs, they are not able to generalize to unseen nodes and therefore are limited. That's why we focus on a more general scenario by training the Graph Neural Networks using the inductive training method. Specifically, given black box access to a graph neural network model that was trained inductively, we are able to predict whether there exists a link between any two nodes of the training graph or not. Our experiments show that there exist efficient ways to properly reveal sensitive information about the training graphs of inductive trained graph neural networks, which leads to big privacy concerns. Depending on data sets and the graph neural network model we achieve up to 0.8955 F1-Score while having an average performance of 0.7817 F1-Score regarding all our attacks.

Acknowledgements

I would like to thank my advisor, Xinlei He and my supervisor, Dr. Yang Zhang for guiding me through this thesis. I really appreciate the time spent for our meetings, friendly talks and messages and preparation talks. Without them I couldn't say, that writing my thesis was the best task during my studies. I learned so much about machine learning in general and specifically GNNs, which wouldn't be possible without them. I would also like to thank Dr. Cristian-Alexandru Staicu for being my second reviewer. Thanks also to the CISPA department for letting me use their resources. Without them this thesis would have taken much longer. I would also want to thank all my friends, who reviewed the thesis for their awesome feedback. Big thanks also to my family and friends who had to put up with me during my studies. It's because of them, that I am now at this point in my life.

Contents

Abstract **v**

Acknowledgements **vii**

1 Introduction **1**
1.1 Motivation 1
1.2 Outline 2

2 Related Work **3**

3 Background **7**
3.1 Neural Networks 7
3.2 Graphs 8
3.3 Graph Neural Networks 9
3.4 Link Stealing Attacks 11

4 Attacks **13**
4.1 Problem Statement 13
4.2 Intuition 13
4.3 Threat Model 14
4.4 Attack Methodology 14

5 Implementation **19**
5.1 Datasets 19
5.2 Target Models 21
5.3 Attack Model 23

6 Evaluation **25**
6.1 Datasets 25
6.2 Metric 25
6.3 Attack Performance 26
6.4 Possible Defense 30
6.5 Summary of Results 31

7 Discussion **33**
7.1 Findings 33

7.2 Impact 34
7.3 Future Work 34

8 Conclusion **35**

List of Figures **36**

List of Tables **39**

Bibliography **47**

Chapter 1

Introduction

1.1 Motivation

A graph is a data structure that is used to model large data and the relationships between their entities [1, 2]. Graphs consist of nodes and edges and are excellent at representing impossibly large data sets. In a social network for example, the nodes are the users that are registered in the network and the edges represent whether the users know each other or not. If they know each other, the two nodes will be linked in the graph and if they don't know each other, the link between them does not exist. A graph itself can be deemed as intellectual property of the data owner, since one 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 patients in healthcare-analytic data sets. 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 tasks to directly operate on graphs. For example they can be trained to perform node classification tasks[5]. More precisely, given a graph containing a few labeled nodes, the model is trained to predict the labels of the other, unlabeled nodes in the graph by considering the graph structure, feature vectors or links of the nodes. They can also be used to perform link prediction, where GNNs are trained to predict whether two nodes are linked or not. In social networks this is called friendship prediction [6].

A Graph Neural Network can be trained in different ways, depending on the purpose it will be used for in the future. One way is to train them using the transductive setting [7–10]. Therefore, we consider the graph to be fixed. Meaning that neither the edges nor

the feature vectors of the nodes change during the lifetime of the graph. Regarding the node classification task that means, that there are some labeled nodes, which are used for training the GNN, and many unlabeled nodes, which need to be classified correctly. Nevertheless this training method is possible, it hardly can be applied to real world problems like training a graph neural network on social network data. That's why graphs in most cases keep evolving. E.g. in social networks, every day new users register to the network while others delete their accounts. To address this problem GNNs can also be trained using the inductive setting [11–13]. More specifically, instead of providing a fixed graph as input and training the GNN to learn the local graph structure, we now want the model to learn an aggregation and update function. These two functions are used to update a nodes feature vector with the aggregation of its neighborhoods feature vectors. In that way, only a partial graph is used for training the graph neural network instead of considering the full graph structure. With the inductive setting, the trained model can better generalize to unseen nodes, by aggregating their neighborhood, updating the nodes feature vector and querying the GNN on the updated result. Now it is possible to update the GNN on new nodes without retraining it completely.

In our work, we want to show, that inductive trained graph neural networks are very likely to leak sensitive information about their training graph. Meaning that queries on a partial graph of the training graph can reveal links, that are deemed confidential and thus lead to a big privacy risk.

1.2 Outline

This thesis starts giving an overview of machine learning and some privacy breaching attacks, like membership inference attacks or model inversion attacks, that have been developed over the past years. Especially considering graphs and graph neural networks we take a first look at link stealing attacks that have been proposed the first time in year 2020. We will then provide some background information about graphs, neural networks, graph neural networks and link stealing attacks. We describe the intuition behind those concepts, where and why we use graph neural networks and what the difference between transductive and inductive training is. After that we propose our attacks against inductive trained graph neural networks, which aim to steal links from the target models training graph, talking about their intuition, their functionality and implementation. At the end, we will evaluate our findings by presenting our experimental results and discuss their impact. We also provide some ideas about possible defenses and future work, that might be interesting.

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 Deep Neural Networks began to dominate different domains in year 2012, attacks against these models were also found and further developed [16, 17]. Today it is well known, 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 investigated, 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 imitates the behaviour of the target. Tramèr et al. [32] proposed simple model extraction

attacks, which were able to steal target models with near-perfect fidelity. A similar approach was proposed by Wang and Gong [33], who were able to successfully steal the hyper parameters 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 perturbations 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, 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 under laying training graph [44]. They call their attacks *Link Stealing Attacks*. Given black box access to a transductive trained 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. Basically, considering a GNN that was trained using the transductive setting to perform a node classification task, the prediction of the model for each node is fixed, since it only can handle fixed graphs. However, when the GNN is trained inductively, it is able to generalize to unseen nodes, leading to the possibility of different prediction results, based on the amount of information, the graph provides. In our work we consider inductive trained Graph Neural Networks and explore their

vulnerability against *Link Stealing Attacks*. Specifically, given black box access to an inductive trained Graph Neural Network, we aim to predict whether there exists a link between any two nodes of a graph, that was used for training.

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.

The simplest class of Neural Networks are Feed Forward Neural Networks. Like other classes, they consists of multiple layers. An input layer, one or 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 a fully connected neural network is called a multilayer perceptron (MLP).

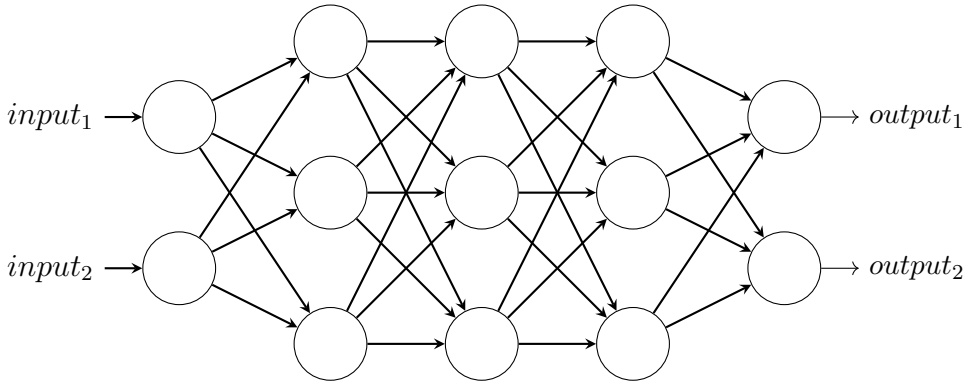

Figure 3.1: Multilayer Perceptron

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 aim to learn how to solve the given 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 phase, 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 $\overrightarrow{a}$ 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, workplace 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 exist a known relationship between them ($e_{EC}, e_{CE} \notin E$).

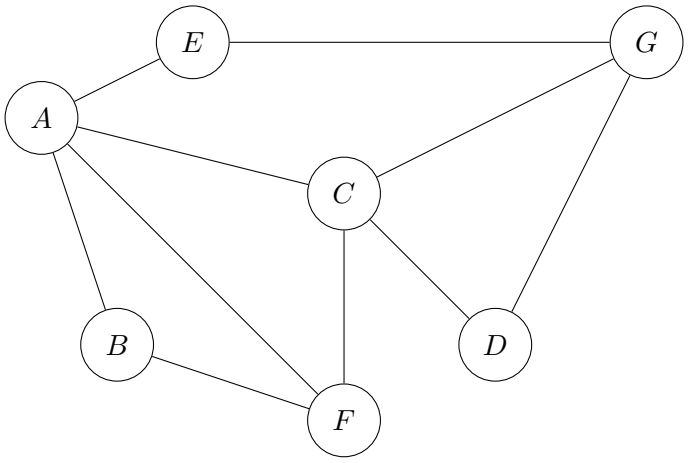

Figure 3.2: Undirected Graph

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 $\overrightarrow{a}$ of user a contains all information the company already has regarding this user. That could be the 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 $v_{workplace} = \{v \mid \forall v \in V : v\text{'s } workplace \text{ is known}\}$, the company knows the workplace from and some users $v_{unknown} = \{v \mid \forall v \in V : v\text{'s } workplace \text{ is unknown}\}$, where the attribute is unknown. Based on the information G 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 missing attribute *workplace* is the label. This example is one of many tasks a Graph Neural Network can perform. Others could be graph classification, where f should be able to predict whether a given graph is a subgraph of another one, or link prediction, which is used for friendship prediction in social networks. 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.

Transductive Setting

In the transductive setting [5] we consider the graph being fixed. Meaning, that neither the links nor the feature vectors of the nodes change over the time. Furthermore, the complete graph is provided as input to a Graph Neural Network model f, including

all nodes, their links and their feature vectors. Thus, the model learns hidden layer representations that encode both local graph structure and features of nodes to perform its task, by aggregating information throughout the graph. In the example given above, we want to predict the *workplace* of unlabeled users. Since this setting operates on the full graph, we consider the information the graph provides as known all the time, especially during the training phase. This implies, that all feature vectors are visible for f during training. Meaning, that if new users join the network, f needs to be retrained because the graph changed, making it really hard to apply this setting to real world problems. Datasets like social networks change every day, such that a model which is trained transductive, must be retrained as often as the structure changes, to maintain their accuracy. This leads to high computational costs and time constraints and thus is not very practical for very large, constant changing datasets. Based on this problem another learning method can be used, which does not use the full graph for training but uses partial graphs instead.

Inductive Setting

For inductive learning, we do not provide the full graph as input for the Graph Neural Network model f. Instead an aggregation function $aggregate(n)$ is used, which aggregates the feature vectors of the k-hop neighborhood of the node n to obtain the input for f. We can set k to any number. $k = 0$ does only consider n's feature vector as input and no aggregated neighborhood embeddings. $k = 1$ aggregates the feature vectors of n's neighbors with n's own feature vector, while $k = 2$ also considers the neighbors of the neighbors of n. Besides this aggregation function the model also learns an update function $update(n)$, which updates n's feature vector with the aggregation of it's neighborhood. In that way $\overrightarrow{n}$ not only contains n's features but also the information of it's neighbors. Thanks to this algorithms, it's no longer necessary to train f on the whole graph. To better understand the process, we continue with the example given above. Lets assume that three new users register to the social network, nobody providing their 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. This happens, by aggregating their neighborhood feature vectors with the own one and querying f on the updated feature vector. f will then predict the workplace based on the provided aggregation, since it was trained to find patterns in vectors that lead to certain predictions. Therefore, inductive trained Graph Neural Networks are able to generalize to unseen nodes, making them easier to use for real world problems.

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 more widely used compared to the transductive learning method.

3.4 Link Stealing Attacks

Link Stealing Attacks have been introduced the first time by He et al. [44]. Given a Graph Neural Network, they performed different attacks with different attack methodologies to steal links from the graph that was used for training the target GNN.

Let f be the target Graph Neural Network model, that was trained on some graph dataset $G = (V, E)$. We assume the adversary was able to obtain some partial graph of G and define it as $G_s = (V_s, E_s)$. Furthermore, we assume that E_s is incomplete. This means, that some links that originally have been in E are missing in E_s. The goal of link stealing attacks is to recover these missing edges. To achieve the reconstruction, the target model will be queried on two nodes $i, j \in V_s$. Since f was trained to perform some task, the outputs $f(i)$ and $f(j)$ will be some posterior representing f's prediction. Based on those posteriors, an adversary will train an attack model A, that infers, whether the two nodes i and j have originally been connected in G, and therefore the edge e_{ij} is missing in E_s, or if they haven't been linked in the first place. To train the attack model, one need some positive and some negative samples. We define the positive samples $pos = \{(i, j) \mid i, j \in V_s : e_{ij} \in E \wedge e_{ij} \notin E_s\}$ as set of node pairs, that have been connected in G but aren't connected in G_s. The negative samples $neg = \{(i, j) \mid i, j \in V_s : e_{ij} \notin E \wedge e_{ij} \notin E_s\}$ are defined as set of node pairs, that haven't been connected in neither of the two graphs. As input for A the authors used the concatenation of the two posteriors or a vector containing the results of eight common distance functions, to describe the similarity of the two posteriors $f(i)$ and $f(j)$. The basic intuition is the following: The posterior outputs $f(i)$ and $f(j)$ should be more similar, if the two nodes i and j have been connected in G. If they haven't been connected in G, then f would output two posteriors, that are less similar.

By completing G_s, an adversary steals sensitive information about the training graph of the target model f, since G_s is a partial graph of f's training graph G.

He et al. performed their attacks on transductive trained Graph Neural Networks and achieved high accuracy in recovering the links of G_s. While the authors used GNNs that only can handle fixed graphs, in our work, we focused on link stealing attacks on inductive trained Graph Neural Networks. Using the inductive training setting, our

models are able to generalize to unseen nodes, leading to different model predictions based on different information the graph provides. By the end of this thesis however, we will show, that inductive trained GNNs are very likely to reveal sensitive information about their training graph as well.

Chapter 4

Attacks

In recent work He et al. [44] proposed the first link stealing attacks on Graph Neural Networks. They focused on transductive trained Graph Neural Networks and were able to steal links of the graph, that was used for training the given target model. In our work, we want to show, that it is possible for an adversary to steal links from the training graph, given black-box access to an inductive trained target Graph Neural Network model.

4.1 Problem Statement

Let f_t be the target Graph Neural Network model, trained on a graph G, to perform some machine learning task. Let $G_s = (V_s, E_s)$ be a subgraph of G with $|V_s|$ nodes and $|E_s|$ edges. We assume, that some of G_s's links/edges are missing. Meaning for two nodes $u, v \in V_s$, the link (u, v) exists in the training graph G but is missing in G_s. The goal of an adversary A is to recover the missing links from G_s. More precisely, the adversary wants to infer whether $e_{uv} \in G \wedge e_{uv} \notin G_s$ (A recovered the missing link between u and v) or $e_{uv} \notin G \wedge e_{uv} \notin G_s$ (the link between u and v 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: $post_i = f_t(i)$ and $post_j = f_t(j)$. A can be trained based on the similarity of the two posteriors, when i and j originally have been connected and the edge is missing in G_s or how similar they are when there is no edge to be recovered.

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 a shadow Graph Neural Network model, which was trained by the adversary using a shadow 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 the shadow model f_A to sample D_A. We assume, that in any attack G_s is a subgraph of D_{f_t} and G_A is a subgraph of D_{f_A}. Meaning, that G_s and D_{f_t} share the same dataset distribution as well as G_A and D_{f_A}. 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 $D_{f_t} = D_{f_A}$ and that G_A is a subgraph of D_{f_t}. Table 8.2 can be used to look up notation descriptions.

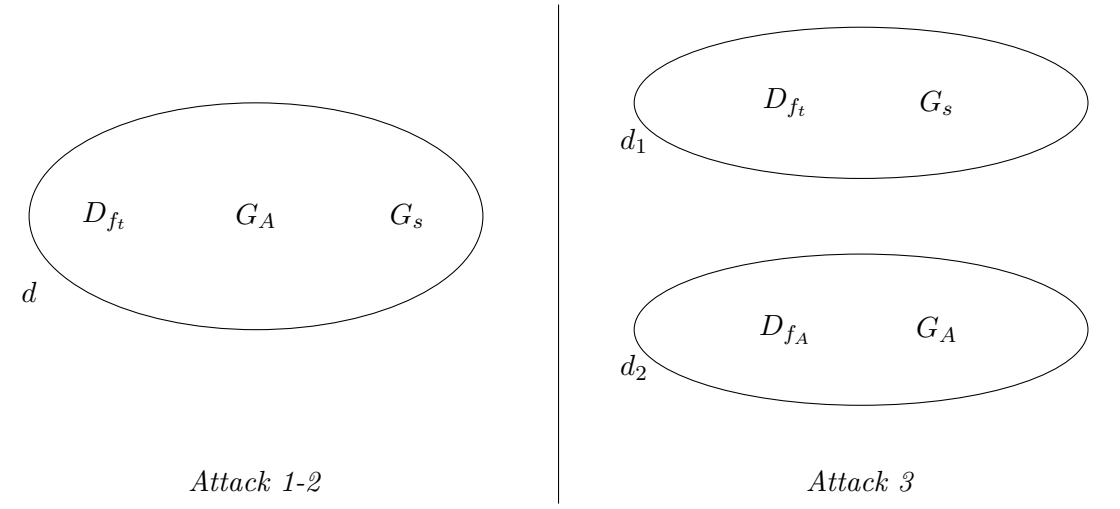

Figure 4.1: Dataset Distributions for Link Stealing Attacks

4.4 Attack Methodology

Let f_t be the target Graph Neural Network model and G_s a subgraph of D_{f_t}. We assume that G_s is not complete. More precisely, there exist edges (i, j) between any nodes $i, j \in G_s$, with $(i, j) \in D_{f_t}$ but $(i, j) \notin G_s$. The adversary A queries f_t on both nodes i and j, obtaining the posterior output of the target model $post_i = f_t(G_s, i)$ and $post_j = f_t(G_s, j)$.

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 the target model f_t on the partial graph G_A. That can be done, because $|post_u| = |f_t(u)|$ (training phase) and $|post_v| = |f_t(v)|$ (attack phase), with $u \in G_A$ and $v \in G_s$, 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 in the training graph D_{f_t} and the edge is missing in G_s or not.

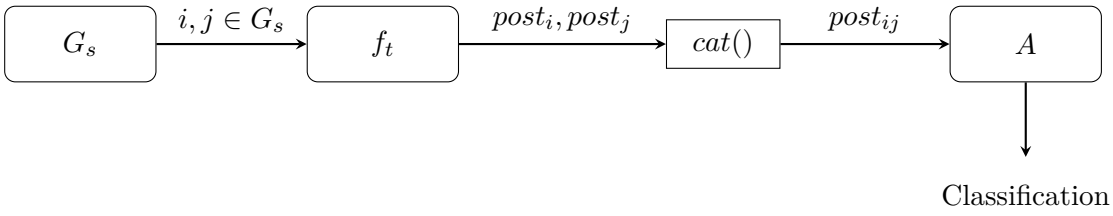

Figure 4.2: Flow Chart - Attack 1

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 (*Attack 3*) we 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 in the training graph D_{f_t} and the edge is missing in G_s or not.

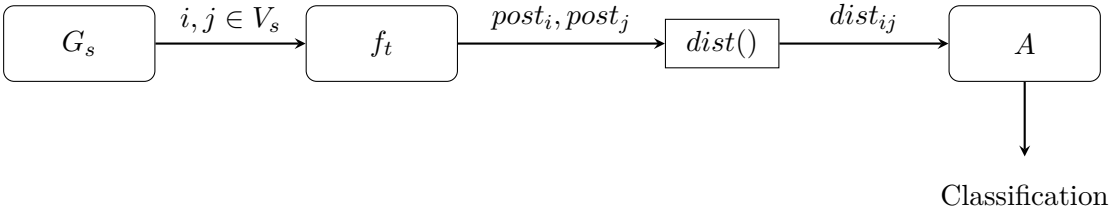

Figure 4.3: Flow Chart - Attack 2

Attack 3

In *Attack 3* we consider G_A and D_{f_t} from different dataset distributions. So the adversary trains a shadow Graph Neural Network model f_A with a shadow 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 $|post_u| = |f_A(u)|$ (training phase) and $|post_v| = |f_t(v)|$ (attack phase), with $u \in G_A$ and $v \in G_s$, may have different dimensions. Based on this assumption, we need to sample the input for A, like it was done in *Attack 2*, by creating features based on the posteriors, instead of using them directly, like it was done in *Attack 1*. As features we again use the eight distance metrics, to measure the distance between $post_i$ and $post_j$. So we construct the input $dist_{ij}$ for A as $dist_{ij} = dist(post_i, post_j)$. Given $dist_{ij}$, A now can infer, whether i and j have been connected in the training graph D_{f_t} and the edge is missing in G_s or not.

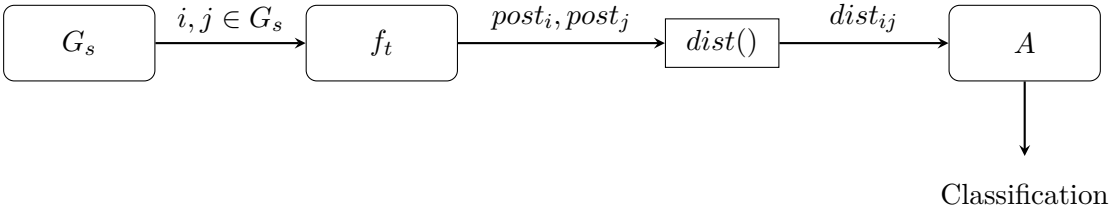

Figure 4.4: Flow Chart - Attack 3

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	$inp_{ij} = cat(post_i, post_j)$	$\|inp_{ij}\| = \|post_i\| + \|post_j\| = 2 * \|post_i\|$
Attack 2	Same	$inp_{ij} = dist(post_i, post_j)$	$\|inp_{ij}\| = 8$
Attack 3	Different	$inp_{ij} = dist(post_i, post_j)$	$\|inp_{ij}\| = 8$

Table 4.1: Attack Methodology

Chapter 5

Implementation

In order to analyze how effective our attacks can steal links from the training graph of an inductive trained Graph Neural Network, we performed several experiments by attacking GNNs that have been trained to perform node classification. In this chapter we want to go through the implementation of our attacks. We covered multiple datasets and Graph Neural Network models, leading to an amount of ~200 experiments. Due to 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. All of the datasets we used are from the same domain - Citation Networks. This is because these networks are open source and similar to social networks in their structure. Due to computational constraints we didn't use social network datasets like reddit.

Name	Number of Nodes	Number of Edges	Number of Classes	Feature Amount
Cora	2.708	5.429	7	1.433
CiteSeer	3.327	4.732	6	3.703
Pubmed	19.717	44.338	3	500

Table 5.1: Dataset Information

Sample Datasets for Experiments

In total we train target models on all three datasets - *Cora*, *CiteSeer* and *Pubmed* . To train the attack models, we sample an adversary dataset based on the incomplete subgraph.

Same Dataset Distributions - Attack 1-2

Let $D_{f_t} = (V_{f_t}, E_{f_t})$ be one of our three original datasets with $|V_{f_t}|$ nodes and $|E_{f_t}|$ edges. We denote $D' = (V', E')$ as subgraph of D_{f_t}. 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.

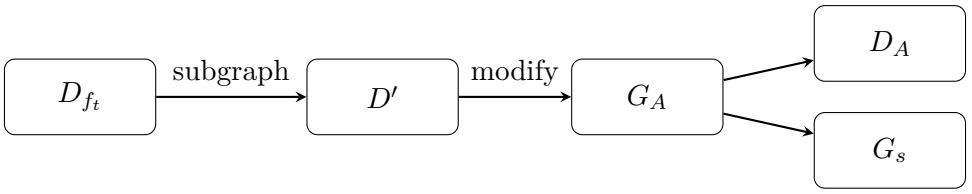

Figure 5.1: Sampling Datasets - Same Dataset Distribution - Attack 1-2

We define $G_A = (V_A, E_A)$ with $V_A = V'$ and $E_A = E'$, which is now an exact copy of D'. To sample D_A, we first collect a set of positive samples $pos = \{(i, j, 1) \mid \forall i, j \in V' : (i, j) \in E' \wedge |pos| < ((1 - \alpha) * |E'|)\}$, containing pairs of nodes, that are connected in D', where α denotes the percentage of known edges. Now, we collect a set of negative samples $neg = \{(i, j, 0) \mid \forall i, j \in V' : (i, j) \notin E' \wedge |neg| < ((1 - \alpha) * |E'|)\}$, containing pairs of nodes, that are not connected in D'. We then delete all edges we sampled for pos, in our graph clone G_A, to simulate the missing edges, we want to steal. This leads to $E_A = \{(i, j) \mid \forall (i, j) \in E' : (i, j) \notin pos\}$. G_A is now a modified graph that contains less edges then the original graph D' and we define a raw-dataset $raw = pos \cup neg$, containing the positive and negative samples obtained from D'. As the next step, we create the adversary's dataset $D_A = \{(post_{ij}, l) \mid \forall (i, j, l) \in raw : post_{ij} = concat(f_t(i), f_t(j))\}$ for *Attack 1* and $D_A = \{(dist_{ij}, l) \mid \forall (i, j, l) \in raw : dist_{ij} = dist(f_t(i), f_t(j))\}$ for *Attack 2*. $f_t(i)$ returns the node classification output posterior of the target model, when it is queried on node i. $concat(a, b)$ concatenates the output posteriors a and b with each other returning the feature we will train the attack model on. l denotes the label either being 1 (positive sample) or 0 (negative sample). $dist(a, b) = [Cosine(a, b), ..., Sqeuclidean(a, b)]$, return the vector containing 8 different distance values like described in Section 4.3. With our adversary's dataset D_A we can now continue training our attack model using either

$post_{ij}$ or $dist_{ij}$ as input features and l as class. G_s represents the incomplete subgraph of D_{f_t}, which will be used to test the adversary performance on f_f. More precisely, how well does the adversary can predict correctly whether a link between two nodes in G_s is missing or not.

Different Dataset Distributions - Attack 3

Let D_{f_A} and D_{f_t} be two of our three original datasets. We use D_{f_A} to train the shadow model f_A and D_{f_t} to train our target model f_t. We denote D'_1 as subgraph of D_{f_A} and D'_2 as subgraph of D_{f_t}. To sample D_A we follow the same steps as described in *Attack 1.* Firstly we create an incomplete graph G_A by randomly deleting some edges. We then sample D_A by querying f_A on G_A. To obtain G_s we modify D'_2 by randomly deleting some edges and use it to test the adversary performance on f_t.

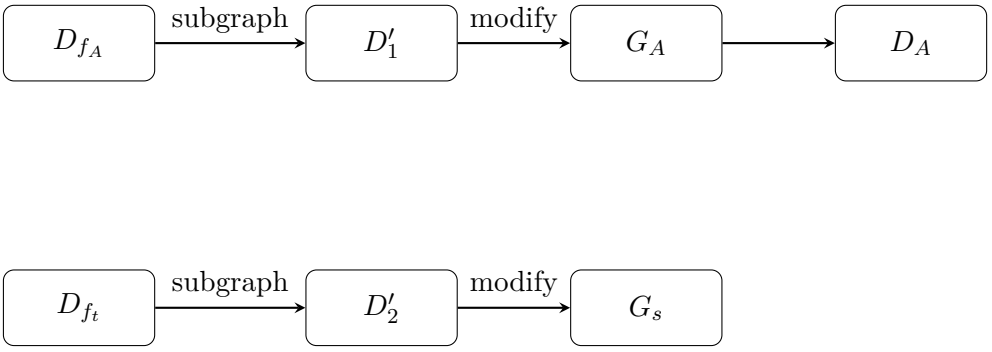

Figure 5.2: Sampling Datasets - Different Dataset Distribution - Attack 3

However, this time the adversary was trained on another dataset distribution than the target model. E.g. the adversary was trained with the *Cora* dataset. We then use A to steal links from the training graph of a target model that was trained on the *CiteSeer* dataset, considering an incomplete subgraph of the *CiteSeer* dataset as G_s. Because of that we use $dist_{ij}$ as input for the attack model instead of the posterior concatenation, like it was done in *Attack 2.*

5.2 Target Models

As our target models, we used three different types of Graph Neural Network models and trained them to perform a node classification task. Therefor, we trained them on our three original datasets Cora, CiteSeer and Pubmed.

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.

Our graphsage target models are trained to perform a node classification task using the *mean* aggregator type. In total we train three graphsage models - one for each dataset. The input layer contains as much neurons as the given datasets' samples provide features. Then, two hidden layers follow, each with 16 neurons. The number of output neurons is equal to the classes the dataset provides. We train each of them for 200 epochs with a learning rate of 0.01, dropout of 0.5 and the Adam optimizer, achieving an average accuracy of 0.809 for *Cora*, 0.746 for *CiteSeer* and 0.872 for *Pubmed*.

Graph Attention Networks

In October 2017 Velickovic et al.[46] presented Graph Attention Networks (GATs), novel neural network architectures that operate directly on graphs. Without requiring any kind of matrix operation or knowledge about the local graph structure the authors specify different weights to different nodes in a nodes' neighborhood by stacking multiple layers in which nodes are able to attend over their neighborhoods' feature vectors. Based on this approach, GATs don't only address transductive but also inductive problems and have achieved or matched state-of-the-art results across four established transductive and inductive graph benchmarks.

Our GAT target models are trained to perform a node classification task. In total we train three GAT models - one for each dataset. The input layer contains as much neurons as the given datasets' samples provide features. Then, one hidden layer with 8 neurons follows. The number of output neurons is equal to the classes the dataset provides. We train each of them for 200 epochs with a learning rate of 0.005, dropout of 0.6 and the Adam optimizer, achieving an average accuracy of 0.778 for *Cora*, 0.609 for *CiteSeer* and 0.891 for *Pubmed*.

Graph Convolutional Networks

In February 2017 Kipf et al. [47] proposed a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. Nevertheless, the authors only consider the transductive setting in their paper, the idea of the algorithm stays the same, when using it for inductive learning. To aggregate the neighborhood, the authors use a single weight matrix per layer and deal with varying node degrees through an appropriate normalization of the adjacency matrix. However, we use a GraphSAGE model with the *gcn* aggregator type to simulate inductive trained graph convolutional networks.

Our GCN target models are trained to perform a node classification task. In total we train three GCN models - one for each dataset. The input layer contains as much neurons as the given datasets' samples provide features. Then, two hidden layers follow, each with 16 neurons. The number of output neurons is equal to the classes the dataset provides. We train each of them for 200 epochs with a learning rate of 0.01, dropout of 0.5 and the Adam optimizer, achieving an average accuracy of 0.817 for *Cora*, 0.748 for *CiteSeer* and 0.861 for *Pubmed*.

The following table provides an overview of our target models and their accuracies regarding their training datasets. Each of them has been trained for 200 epochs,with the Adam optimizer, a learning rate of 0.01 (GraphSAGE and GCN) or 0.005 (GAT) and a dropout of 0.5 (GraphSAGE and GCN) or 0.6 (GAT).

Target Model	Dataset	Accuracy
GraphSage	Cora	0.809
	CiteSeer	0.746
	Pubmed	0.872
GAT	Cora	0.778
	CiteSeer	0.609
	Pubmed	0.891
GCN	Cora	0.817
	CiteSeer	0.748
	Pubmed	0.861

Table 5.2: Target Model Accuracies

5.3 Attack Model

As our attack model we use a Multilayer Perceptron 3.1. Depending on the attack, the MLP has eight (*Attack 2* and *Attack 3*) or two times the size of the posterior output of f (*Attack 1*) input neurons. Then two hidden layers with 16 neurons each follow. The

final output layer consists of 2 neurons, representing the two cases of the two target nodes originally being connected or not. We trained our attack models on the obtained datasets D_A for 200 epochs, with a learning rate of 0.01 and dropout of 0.5 using the Adam optimizer.

Chapter 6

Evaluation

6.1 Datasets

In total we used 3 different Datasets (Section 5.1), which are common used benchmark datasets for evaluating Graph Neural Networks [47–49]. All of them are citation datasets with nodes being publications and edges representing citations among these publications. Furthermore do all datasets contain nodes' attributes and labels.

The method to sample the attack dataset D_A, described in Section 5.1, follows the common practice in the literature of link prediction [50, 51].

6.2 Metric

We use F1-Score as our main evaluation metric. It is a common used metric in binary classification [52–54], since it is the harmonic mean of precision and recall.- The highest value, that is possible, is 1.0, indicating perfect precision and recall. If either the precision or the recall is zero, the F1-Score is 0.0. Leading to f1-Score values between 0 and 1.

Precision

A high precision represents a high probability that the prediction of a Machine Learning model is correct. Let M be a Machine Learning model that was trained to predict whether an email is spam or not. High precision means, that when the model labels an email as malicious, it is correct most of the time and vice versa.

Recall

A high recall represents a high percentage of correctly classified inputs. In the example just given, that means, that M is able to identify a high amount of spam-mails as malicious.

6.3 Attack Performance

The results presented below are the average results of 10 runs. Meaning, that all attacks have been performed multiple times to get a better relation. Note, that the performance varies based on the random choice of the nodes that are included in the partial graph G_s. Each attack is performed on all target models, which have been trained on different datasets. As our baseline we consider that the partial graph G_s doesn't contain any links. We then add 20% of the edges per attack, ending up with 80% known edges.

Attack 1

Like described in Section 4.4 this attack performs link stealing attacks on the same dataset distribution using the concatenation of the posterior outputs of two nodes to infer whether they have been connected or not. Figure 6.1 presents the results for our link stealing attacks on the target models, that have been trained on the CiteSeer dataset. Please find the results for Cora in Figure 8.1 and for Pubmed in Figure 8.2

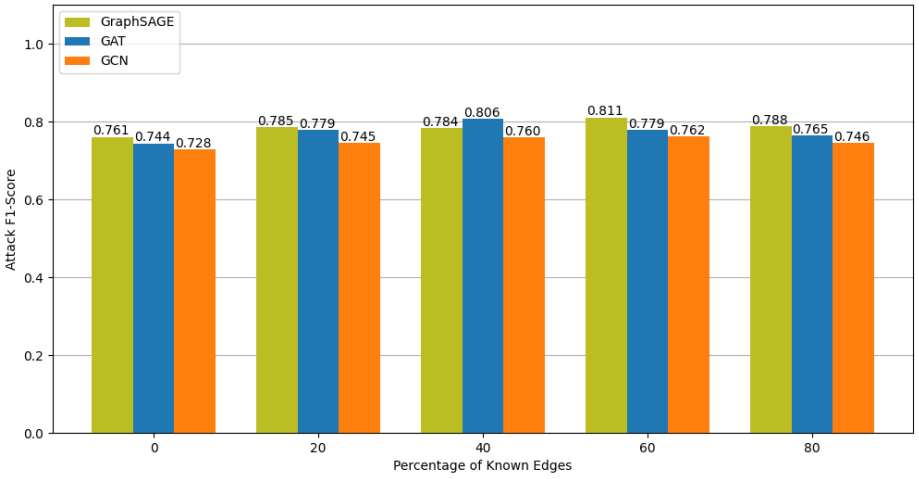

Figure 6.1: Performance of *Attack 1* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the CiteSeer dataset.

Note that our baseline already achieves an average F1-Score of 0.742 depending on Graph Neural Network type and target dataset D_{f_t}. The baseline performed best (0.790 F1-Score) on the Graph Convolutional Neural Network, when it was trained on the CiteSeer dataset and it performed worst (0.666 F1-Score) on the Graph Attention Network when it was trained on the Cora dataset. Intuitively, with rising amount of known edges, we would expect a higher performance of our attacks. However, in some cases, the performance grows until 40% or 60% of known edges and drops afterwards. More precisely, sometimes the attack performance is better with only 60% of known edges than with 80%. The reason we observe this, is the following. Since we use the deleted edges as positive samples for our training data, the more edges are known, the less data will be provided for training the attack model. Leading to more training data in our baseline than in our 80%-known-edges attack. With respect to all performed attacks, the average F1-Score is 0.741. Table 6.1 presents the F1-Scores of our best and worst attacks.

Target Model	D_{f_t}	G_A Distribution	α	F1-Score
GCN	Cora	Cora	0.8	0.811
GraphSAGE	Pubmed	Pubmed	0.2	0.564

Table 6.1: Attack-1: Best and Worst Attack Performance

Attack 2

Like described in Section 4.4 this attack performs link stealing attacks on the same dataset distribution using the calculated distance vector of the posterior outputs of two nodes to infer whether they have been connected or not. Figure 6.2 presents the results for our link stealing attacks on a target model, that was trained on the CiteSeer dataset. Please find the results for Cora in Figure 8.3 and for Pubmed in Figure 8.4

The first noticeable fact is, that using the distance vector instead of the concatenation of the posteriors is more effective. This time the average F1-Score of our baseline is 0.777, again depending on Graph Neural Network type and target dataset D_{f_t}. The baseline performed best (0.837 F1-Score) on the Graph Attention Network, when it was trained on the CiteSeer dataset and it performed worst (0.754 F1-Score) on the GraphSAGE GNN when it was trained on the Pubmed dataset. Again we can observe, that the results support our forecast. With rising amount of known edges, the attack performance grows, leading to an improvement up to 0.08 F1-Score, while comparing the baseline performance with the results of attacks with more known edges. With respect to all performed attacks, the average F1-Score is 0.805. Table 6.2 presents the F1-Scores of our best and worst attacks.

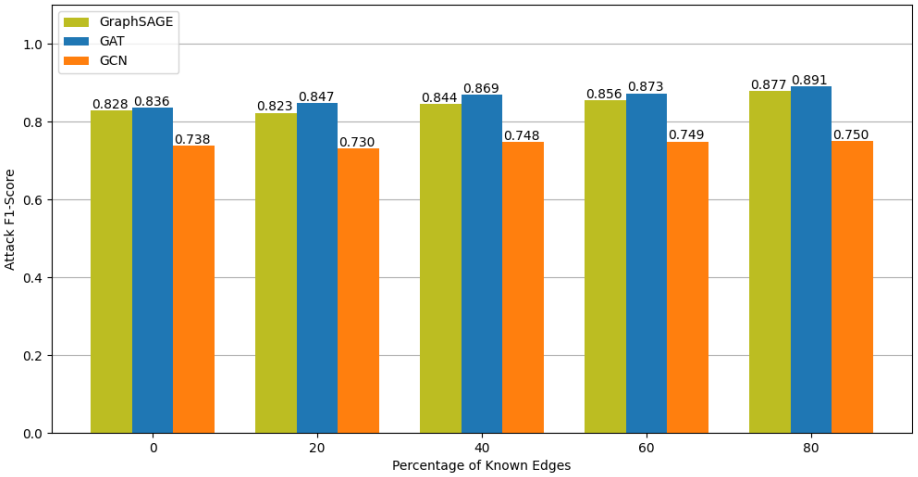

Figure 6.2: Performance of *Attack 2* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the CiteSeer dataset.

Target Model	D_{f_t}	G_A Distribution	α	F1-Score
GAT	CiteSeer	CiteSeer	0.8	0.892
GraphSAGE	Cora	Cora	0.8	0.746

Table 6.2: Attack-2: Best and Worst Attack Performance

Attack 3

Like described in Section 4.4 this attack performs link stealing attacks on a different dataset distribution using the calculated distance vector of the posterior outputs of two nodes to infer whether they have been connected or not. The following Figures 6.3 and 6.4 present the results for our link stealing attacks on a target model, that was trained on the CiteSeer dataset while the adversary was trained on another distribution dataset. Please find the results for Cora in Figures 8.5 and 8.6 and for Pubmed in Figures 8.7 and 8.8.

Evaluating *Attack 3*, we note, that the average attack performance of our baseline (0.756 F1-Score) is higher than the average baseline performance of *Attack 1* but lower than the average baseline performance of *Attack 2*. The results differ, based on the used datasets D_{f_t} and D_A and the architecture of the target model. We achieve a minimum baseline attack performance of 0.627 F1-Score with D_{f_t} being the Cora dataset, the attack model being trained on the Pubmed dataset and the target model being a Graph Convolutional Neural Network. However, when our target model is a Graph Attention Network, which was trained on the CiteSeer dataset, while the attack model was trained on the Pubmed dataset, we can achieve a maximum baseline attack performance of 0.830 F1-Score. Like

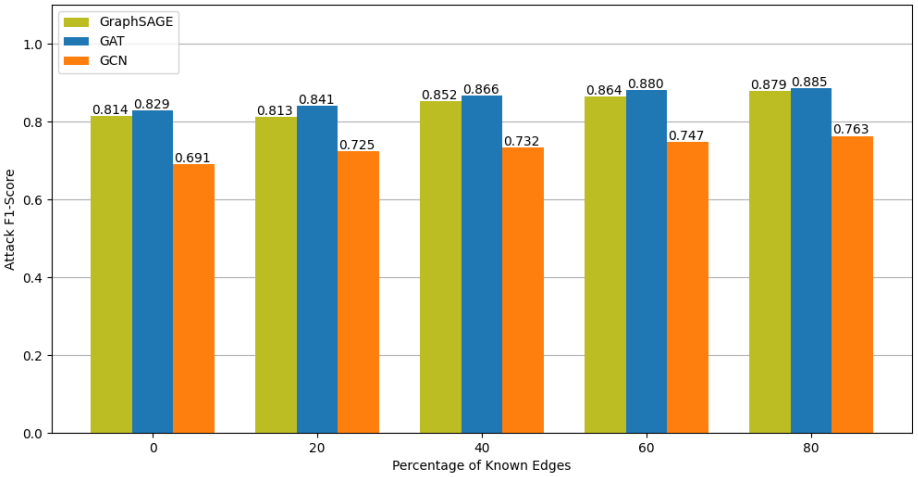

Figure 6.3: Performance of *Attack 3* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the CiteSeer dataset while the shadow model f_A was trained on the Cora dataset.

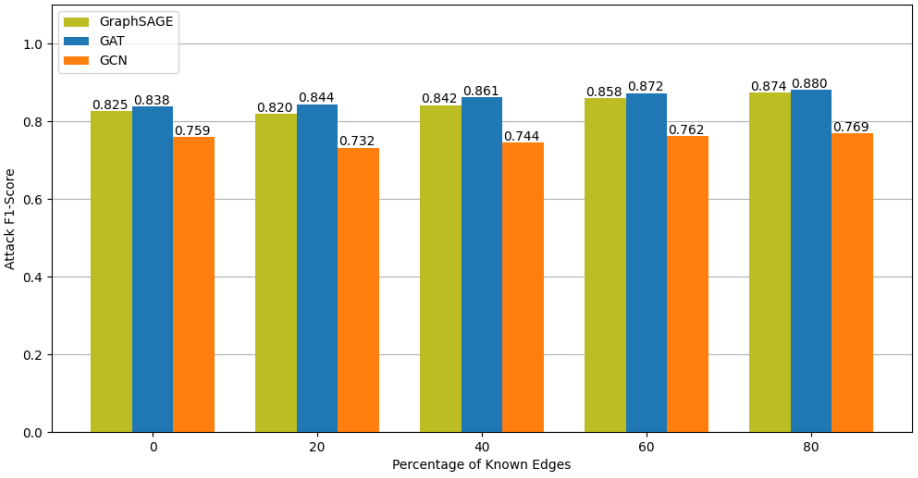

Figure 6.4: Performance of *Attack 3* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the CiteSeer dataset while the shadow model f_A was trained on the Pubmed dataset.

in the other experiments, the attack performance mostly increases, while the amount of known edges rises. This leads to an improvement up to 0.20 F1-Score. With respect to all performed attacks, the average F1-Score is 0.800. Table 6.3 presents the F1-Scores of our best and worst attacks.

The following two tables provide an overview of the best attack performances (Table 6.4)

Target Model	D_{f_t}	G_A Distribution	α	F1-Score
GAT	CiteSeer	Cora	0.8	0.899
GCN	Cora	Pubmed	0.0	0.627

Table 6.3: Attack-3: Best and Worst Attack Performance

and of the average attack performances on our three target model architectures (Table 6.5) with respect to all our three attack types.

Target Model	Attack 1	Attack 2	Attack 3
GraphSAGE	0.758	**0.828**	0.815
GAT	0.802	0.892	**0.899**
GCN	0.811	0.873	**0.878**

Table 6.4: Best Attack Performances on Target Models (F1-Score)

Target Model	Attack 1	Attack 2	Attack 3
GraphSAGE	0.694	0.780	**0.781**
GAT	0.753	**0.825**	0.821
GCN	0.777	**0.810**	0.799

Table 6.5: Average Attack Performances on Target Models (F1-Score)

We notice, that sometimes the performance of *Attack 3* is better than the performance of *Attack 2* (Table 6.4), which is against our intuition, since *Attack 3* uses different distribution datasets. This phenomenon can be explained with the size of G_A. In *Attack 2* G_A is much smaller, meaning that less training data can be provided to train the attack model. Since G_A is a complete dataset - Cora, CiteSeer or Pubmed - in *Attack 3*, the attack model has more data it can be trained on, leading to a better attack performance. In general however (Table 6.5), we observe that *Attack 2* scores better than *Attack 3* on almost all target models.

6.4 Possible Defense

One possible defense, which already has been presented by He et al. [44], is to minimize the posterior output vector of f. Meaning that instead of providing the complete posterior output of f's prediction, f could only present the top k posteriors. In that way the adversary must attack the target model based on less information which makes the attack less effective. Since the performed attacks are very similar and only the architecture and functionality of the target model varies, we can assume, that the defense would lead to a similar drop of attack performance in our work.

However, if we assume, that f always provides the complete output posterior, there still exist some methods to mitigate our attacks. Like He et al. proposed in their work, it is also possible to defend against these attacks by leveraging differential privacy (DP) and adversarial examples. More specifically we could adopt edge-Differential Privacy [55–58]. The approach of Zhang et al. [58] specifies a probability distribution over possible outputs to ensure DP. While it is carefully defined to maximize the utility for the given input, it still provides the required privacy level. Like shown in previous work [59, 60], it is also possible to fool the adversary by adding noise to the prediction of the target model.

6.5 Summary of Results

To sum up, we made the following observations during our experiments. First, our attacks can successfully steal links from inductive trained Graph Neural Networks. For example we were able to steal links from Graph Convolutional Networks with F1-Scores up to 0.896, which shows the effectiveness of our attacks. Second, there exists a dependence between the amount of known edges the adversary has and the attack performance. The more background knowledge the adversary has, the better the results. Furthermore, we also achieve good results with our transferring attack. However, the performance varies dependent on the shadow dataset. But in total the different distribution does not really impact the performance, since the results are similar to same dataset distribution attacks. We observe, that the average attack performance of *Attack 1* scores worst on all our target models. We notice an average improvement of 0.09 F1-Score for GraphSAGE, 0.07 F1-Score for Graph Attention Networks and 0.03 F1-Score for Graph Convolutional Networks, when we compare the concatenation of the posteriors (*Attack 1*) with the sampling of features based on the posteriors (*Attack 2 and Attack 3*). We lastly note, that our attacks perform worst on the GraphSAGE GNN meaning, that these networks seem to be the most resistant ones.

Chapter 7

Discussion

7.1 Findings

In this thesis we wanted to find out, whether it is possible to efficiently steal links from the training graph of a Graph Neural Network, that has been trained inductively. Besides this main question, we also analysed the impact of different distribution datasets on the performance of our attacks and how the choice of features, used to train the attack model, can increase the accuracy of the adversary.

We found that an adversary with black box access to an inductive trained Graph Neural Network is able to efficiently steal links from the target models training graph. To achieve this, the adversary has multiple strategies which differ in the choice of features used to train the attack model and in the dataset distribution of the shadow dataset. We saw, that the attack performance is better, when we sample features on the posterior outputs of the target model instead of using them directly. We therefor compared the concatenation of the posteriors and a distance vector containing values of eight different distance metrics, that measure the similarity between the posteriors, being the input for our attack model. Furthermore we found, that the attack performance is similar when using different dataset distributions to train our attack model. More precisely, we are able to steal links using a transferring attack almost as accurate as using a shadow dataset from the same distribution like the one, the target model was trained on. Therefor we compared the performance of attack models that have been trained on one of our three datasets Cora, CiteSeer or Pubmed while the target model was trained on another dataset, that differs the one the attacker used.

7.2 Impact

Since training data for machine learning models can often be deemed confidential, we note, that stealing links from the training graph of an inductive trained Graph Neural Network leads to a huge privacy risk. The data owner often spends lots of time and resources preprocessing the data and thus can claim the dataset as intellectual property. Furthermore, the training data of a Graph Neural Network often contains sensitive data, that can be revealed using our attacks. This can include health data of patients or sensitive social relationships of social network users. Since nowadays graphs and therefor Graph Neural Networks are getting more and more popular, our attacks have a huge impact on the privacy of the training data. Furthermore, it seems to be hard to protect against our attacks without a tradeoff between good privacy and functionality of the target model.

7.3 Future Work

In future work it would be interesting to see how Link Stealing Attacks perform on target models, that have been trained to perform different downstream tasks. Since all of our target models are trained to perform node classification, it would be interesting to see, whether the task of the Graph Neural Network has some impact on the attack accuracy. Furthermore, one could spend some time figuring out, whether there is a better way of using the raw posteriors as input for the attack model. Since we only concatenate them to one big vector, maybe there is a better way of using them, which leads to better results. Also one could search for even better feature samples than eight distance metrics to use as input for the attack model. Last but definitely not least, one could search for good defenses against our attacks. Since revealing links from the train graph brings huge privacy risks, finding an efficient defense against our attack may be the most important future work.

Chapter 8

Conclusion

Since machine learning and big data are huge points of interest for many people and companies, the need for privacy preserving approaches is indispensable. With our work we want to contribute, by showing privacy risks in common used training processes for Graph Neural Networks. Graphs can be deemed as intellectual property, since the data owner mostly spent much time and resources collecting and preparing the data. However, in most cases graphs also contain highly sensible and confidential data, which makes it even more necessary to ensure privacy. That's why we propose Link Stealing Attacks on inductive trained Graph Neural Networks. Using our attacks, we are able to successfully steal links from a graph that was used to inductively train the target Graph Neural Network model. Only with black box access to the target model an adversary is able to reveal sensitive information about that graph leading to huge privacy concerns. We investigated the performance of our attacks using three types of Graph Neural Networks - GraphSAGE, Graph Attention Networks and Graph Convolutional Networks and trained them on three common used graph datasets - Cora, CiteSeer and Pubmed - to perform node classification. In our experiments we also considered different inputs for our attack model. For some experiments, we concatenate the posterior outputs of our target model, for some others, we sample new features based on the similarity of the posteriors using eight common distance metrics. We showed, that we are able to efficiently infer whether any two nodes in the training graph are linked or not only considering black box access and some shadow dataset. We also showed, that transferring attacks perform with similar results while being trained on different dataset distributions.

To conclude this thesis, we saw that a common used training procedure can lead to significant privacy risks, since the target Graph Neural Network model can be used to reveal sensitive information about the graph that was used for training. We saw that

only black box access and some shadow dataset is enough to perform our attacks and that the performance increases with rising background knowledge.

List of Figures

3.1 Multilayer Perceptron 7
3.2 Undirected Graph 9

4.1 Dataset Distributions for Link Stealing Attacks 14
4.2 Flow Chart - Attack 1 15
4.3 Flow Chart - Attack 2 16
4.4 Flow Chart - Attack 3 16

5.1 Sampling Datasets - Same Dataset Distribution - Attack 1-2 20
5.2 Sampling Datasets - Different Dataset Distribution - Attack 3 21

6.1 Attack 1 - $D_{f_t} = CiteSeer$ 26
6.2 Attack 2 - $D_{f_t} = CiteSeer$ 28
6.3 Attack 3 - $D_{f_t} = CiteSeer$ and $D_A = Cora$ 29
6.4 Attack 3 - $D_{f_t} = CiteSeer$ and $D_A = Pubmed$ 29

8.1 Attack 1 - $D_{f_t} = Cora$ 42
8.2 Attack 1 - $D_{f_t} = Pubmed$ 42
8.3 Attack 2 - $D_{f_t} = Cora$ 43
8.4 Attack 2 - $D_{f_t} = Pubmed$ 43
8.5 Attack 3 - $D_{f_t} = Cora$ and $D_A = CiteSeer$ 44
8.6 Attack 3 - $D_{f_t} = Cora$ and $D_A = Pubmed$ 44
8.7 Attack 3 - $D_{f_t} = Pubmed$ and $D_A = Cora$ 45
8.8 Attack 3 - $D_{f_t} = Pubmed$ and $D_A = CiteSeer$ 45

List of Tables

4.1 Attack Methodology 17

5.1 Dataset Information 19
5.2 Target Model Accuracies 23

6.1 Attack-1: Best and Worst Attack Performance 27
6.2 Attack-2: Best and Worst Attack Performance 28
6.3 Attack-3: Best and Worst Attack Performance 30
6.4 Best Attack Performances on Target Models (F1-Score) 30
6.5 Average Attack Performances on Target Models (F1-Score) 30

8.1 Distance metrics: $f_{t_i}(u)$ represents the i-th component of $f_t(u)$. 41
8.2 Notations 41

Appendix

Metrics	Definition
Cosine	$1 - \frac{f_t(u) \cdot f_t(v)}{\|\|f_t(u)\|\|_2 \, \|\|f_t(v)\|\|_2}$
Euclidean	$\|\|f_t(u) - f_t(v)\|\|_2$
Correlation	$1 - \frac{(f_t(u) - \overline{f_t(u)}) \cdot (f_t(v) - \overline{f_t(v)})}{\|\|(f_t(u) - \overline{f_t(u)})\|\|_2 \|\|(f_t(v) - \overline{f_t(v)})\|\|_2}$
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	$\|\|f_t(u) - f_t(v)\|\|_2^2$

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}	Data set used to train f_t
A	Adversary
D_A	Data set 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	Shadow GNN Model, that is involved in training A
D_{f_A}	Data set used to train f_A
G_s	Incomplete Graph, A performs link stealing attacks on

Table 8.2: Notations

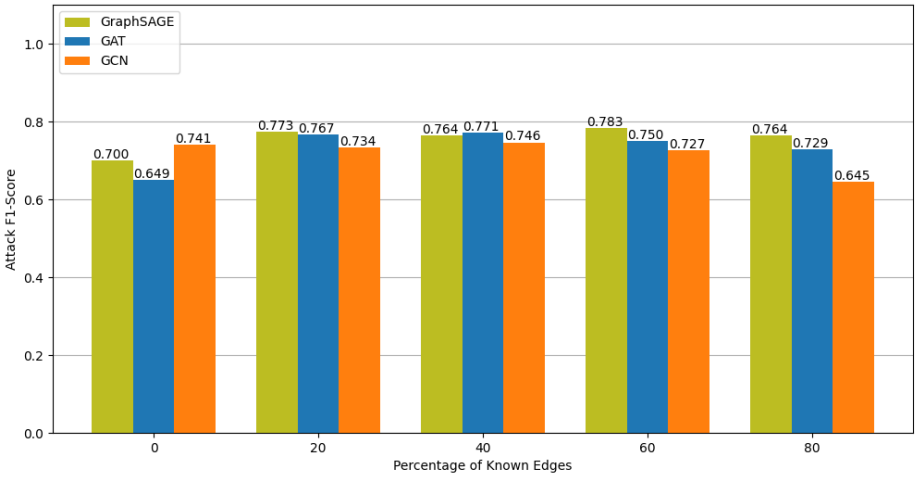

Figure 8.1: Performance of *Attack 1* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the Cora data set.

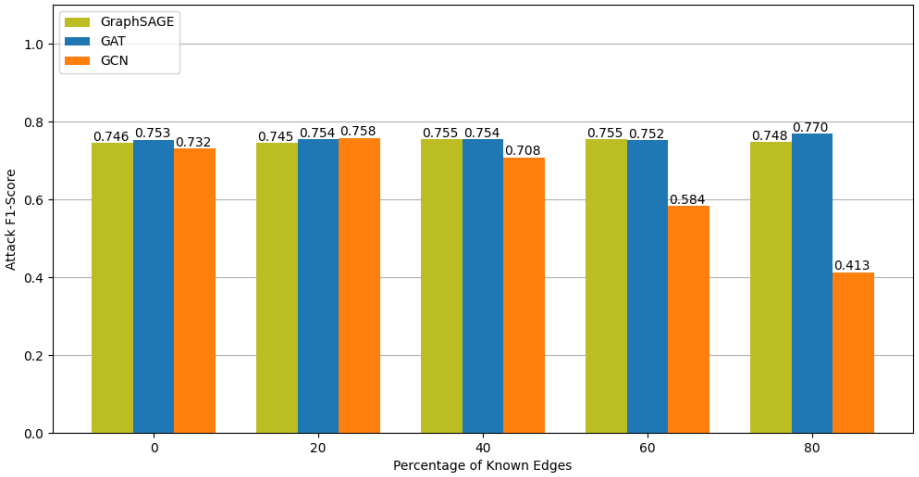

Figure 8.2: Performance of *Attack 1* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the Pubmed data set.

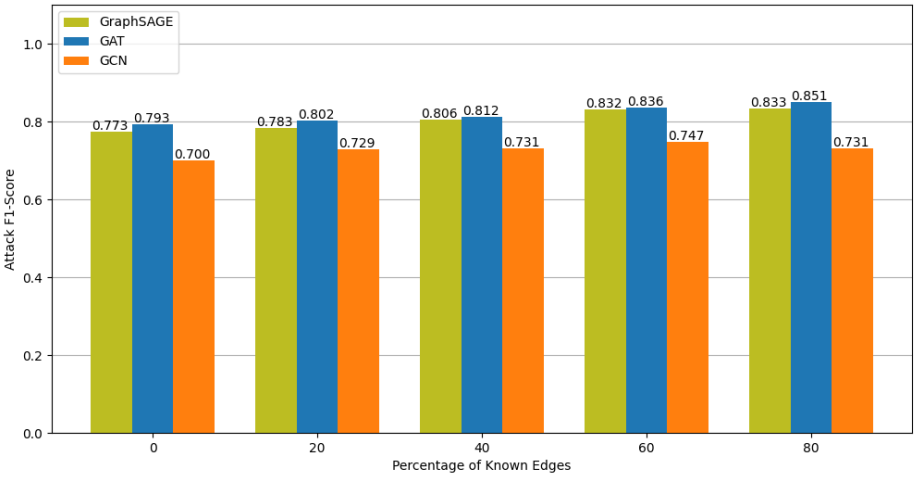

Figure 8.3: Performance of *Attack 2* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the Cora data set.

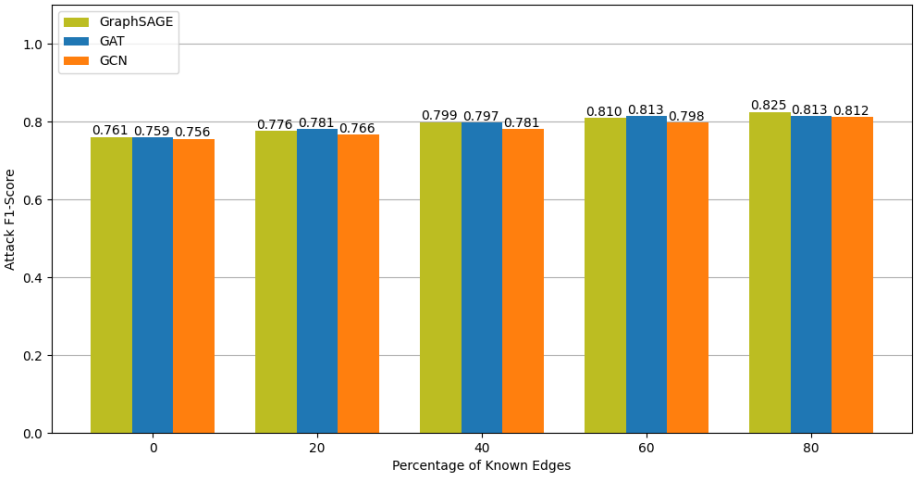

Figure 8.4: Performance of *Attack 2* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the Pubmed data set.

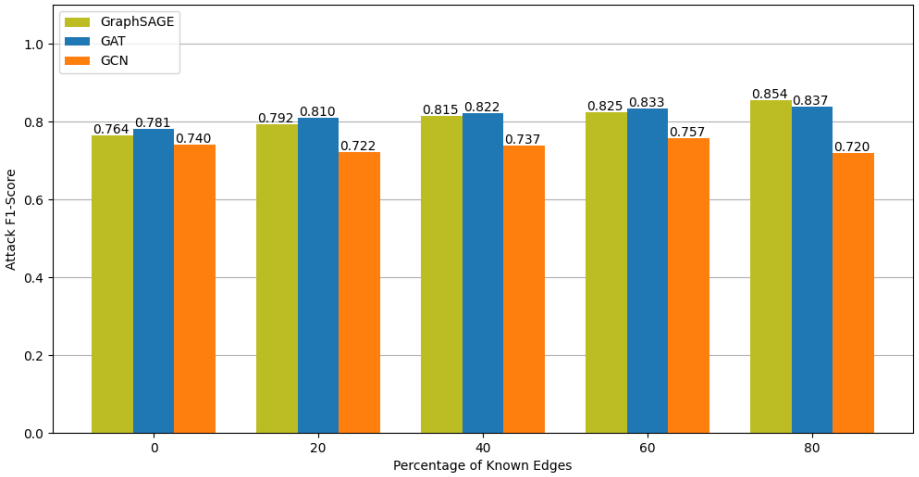

Figure 8.5: Performance of *Attack 3* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the Cora data set while the shadow model f_A was trained on the CiteSeer data set.

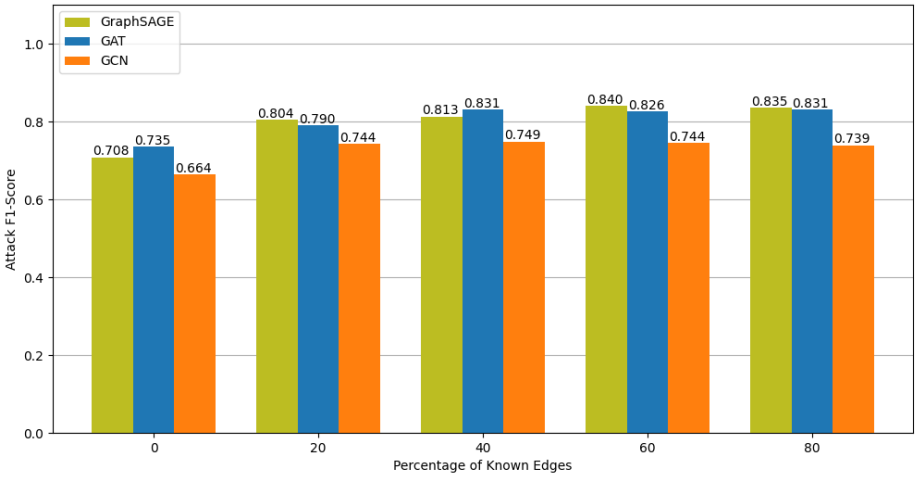

Figure 8.6: Performance of *Attack 3* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the Cora data set while the shadow model f_A was trained on the Pubmed data set.

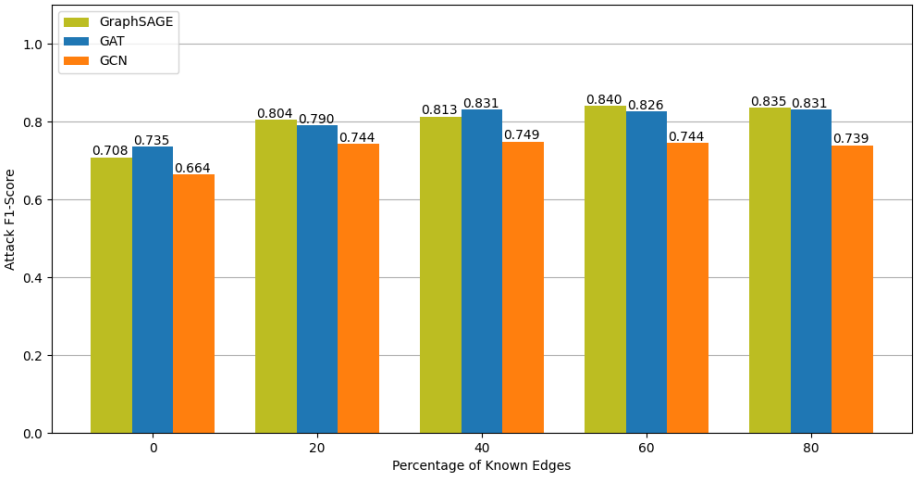

Figure 8.7: Performance of *Attack 3* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the Pubmed data set while the shadow model f_A was trained on the Cora data set.

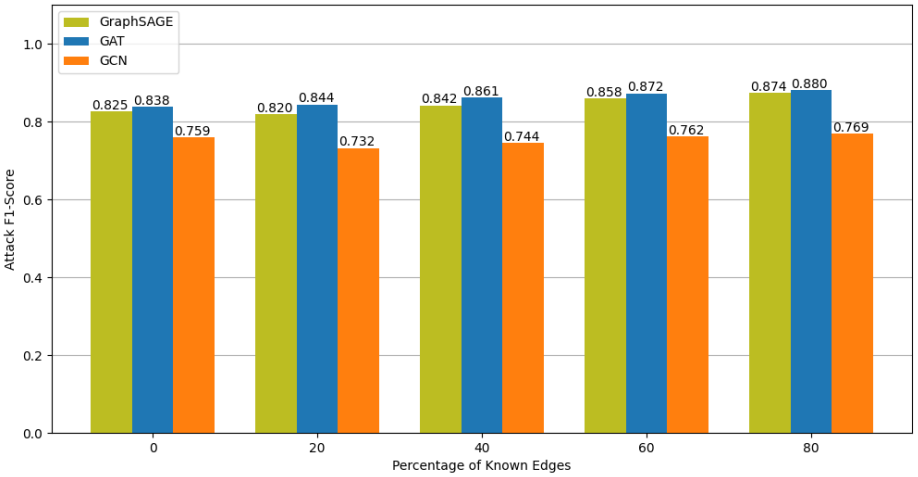

Figure 8.8: Performance of *Attack 3* in F1-Score (y-axis) on our three GNN architectures with rising amount of known edges (x-axis). The target model f_t was trained on the Pubmed data set while the shadow model f_A was trained on the CiteSeer data set.

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.

[46] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

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

[48] D. B. Acharya and D. H. Zhang, "Feature selection and extraction for graph neural networks," 2019.

[49] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graphnas: Graph neural architecture search with reinforcement learning," 2019.

[50] M. Backes, M. Humbert, J. Pang, and Y. Zhang, "walk2friends: Inferring social links from mobility profiles." in *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 1943–1957.

[51] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016.

[52] Z. C. Lipton, C. Elkan, and B. Narayanaswamy, "Thresholding classifiers to maximize f1 score," 2014.

[53] E. Santus, A. Lenci, T.-S. Chiu, Q. Lu, and C.-R. Huang, "Nine features in a random forest to learn taxonomical semantic relations," 2016.

[54] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Predicting domain generation algorithms with long short-term memory networks," 2016.

[55] M. Hay, C. Li, G. Miklau, and D. Jensen, "Accurate estimation of the degree distribution of private networks."

[56] Z. Lu and H. Shen, "Protect edge privacy in path publishing with differential privacy," 2020.

[57] M. Du, K. Wang, Z. Xia, and Y. Zhang, "Differential privacy preserving of training model in wireless big data with edge computing," *IEEE Transactions on Big Data*, vol. 6, no. 2, pp. 283–295, 2020.

[58] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao, "Private release of graph statistics using ladder functions," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, may 2015. [Online]. Available: https://doi.org/10.1145%2F2723372.2737785

[59] J. Jia and N. Z. Gong, "Attriguard: A practical defense against attribute inference attacks via adversarial machine learning," 2020.

[60] J. Jia, A. Salem, M. Backes, Y. Zhang, and N. Z. Gong, "Memguard: Defending against black-box membership inference attacks via adversarial examples," 2019.