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 inductive trained graph neural networks can reveal sensitive information about their training graph. We focus 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 setting. 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 the model, are linked or not. In our work, we now focus on the inductive setting. Specifically, given a black box access to a graph neural network model that was trained inductively, we aim to predict whether there exists a link between any two nodes of the training graph or not. present results

Acknowledgements

Contents

Abstract **v**

Acknowledgements **vii**

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

2 Related Work **3**

3 Background **5**
3.1 Neural Networks 5
3.2 Graphs 6
3.3 Graph Neural Networks 7
3.4 Link Stealing Attacks 9

4 Attacks **11**
4.1 Adversary's Goal 11
4.2 Intuition 11
4.3 Threat Model 12
4.4 Attack Methodology 12

5 Implementation **15**
5.1 Datasets 15
5.2 Target Models 17
5.3 Attack Model 19

6 Evaluation **21**
6.1 Datasets 21
6.2 Metric 21
6.2.1 Precision 21
6.2.2 Recall 22
6.3 Attack Performance 22
6.4 Possible Defense 22
6.5 Summary of Results 22

7 Discussion **23**
7.1 Findings 23
7.2 Impact 23
7.3 Future Work 23

8 Conclusion **25**

List of Figures **25**

List of Tables **29**

Bibliography **33**

Chapter 1

Introduction

1.1 Motivation

Update: Not all inductive settings use aggregation functions Transductive (use full graph) - Inductive (use Local graph / neighborhood)

A graph is a data structure 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 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 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 directly 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 later. One way is to train them transductive [7–10]. In the transductive setting, 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 trained model. Regarding the node classification problem that means, that we have some labeled nodes, we use for training, and many unlabeled nodes, we want to classify correctly. Nevertheless this training method is possible theoretically, it hardly can be applied to real world problems like training on social networks. That's why graphs in most cases keep evolving. E.g. in social networks, every day new users register and others delete their accounts. For datasets like that GNNs can also be trained inductive [11–13]. Specifically, instead of providing the complete graph as input and train the model to learn the local graph structure, we now want the model to learn an aggregation and update function. These functions are used to update a nodes feature vector with the aggregation of its neighborhood. In that way, only a partial graph is used for training the model instead of using the full graph. With the inductive setting, the model can generalize to unseen nodes, by aggregating their neighborhood, updating the nodes feature vector and querying the model on the updated result. In that way it is now possible to update the model on new nodes without retraining it over and over again.

In our work, we 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 become big privacy concerns.

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 training graph [44]. They call their attacks *Link Stealing Attacks*. Given a 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. Specifically, given a 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.

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 a fully connected neural network is called a multilayer perceptron (MLP).

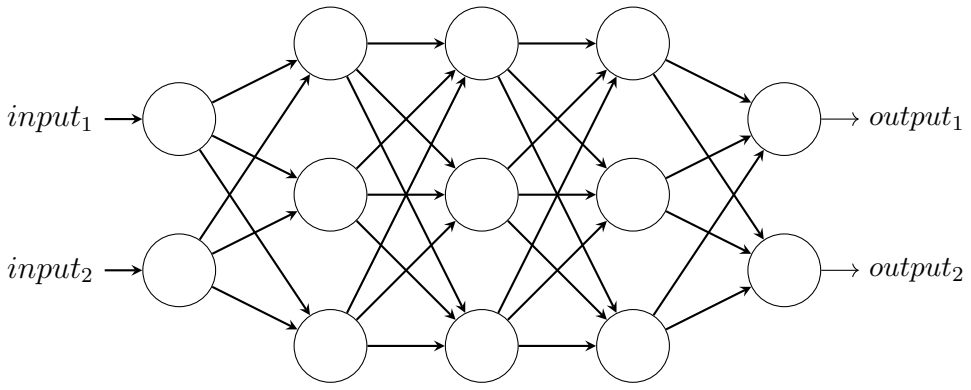

Figure 3.1: Multilayer Perceptron

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 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 exists a known relationship between them ($e_{EC}, e_{CE} \notin E$).

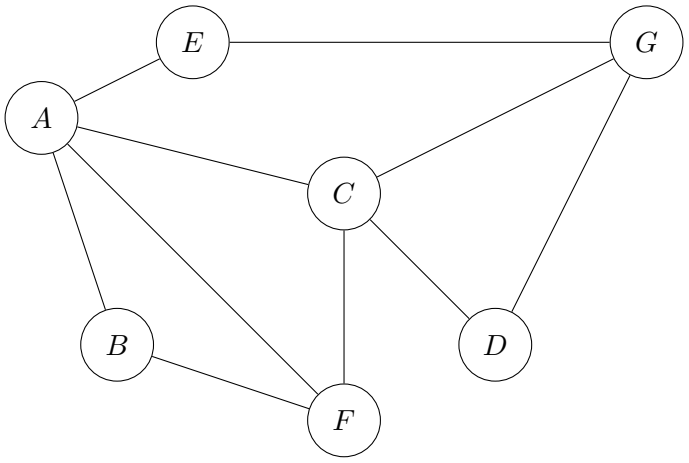

Figure 3.2: Undirected Graph

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 $\overrightarrow{a}$ 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 $v_{workplace} = \{v \mid \forall v \in V : v \text{ has } workplace \text{ given}\}$, 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 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. Other options 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 not the feature vectors of the nodes change. 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 provide the full graph as input for the model. Instead an aggregation function $neighborhood(n)$ 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 neighbor embeddings. $k = 1$ aggregates the neighbor vectors of n with n's 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 but instead only learn and later on apply these two functions. To better understand the process, we continue with the example given above. 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. 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. Therefor, 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 commonly used, while the transductive one isn't.

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.

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_s are missing. 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 therefor 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 $pos = \{(i, j) \mid i, j \in V_A : e_{ij} \in E \wedge e_{ij} \notin E_s\}$ 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. In our work, we focus on link stealing attacks on inductive trained graph neural networks and we hope to show, that they 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 Adversary's Goal

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) does exist in the training graph G but is missing in G_s. The goal of an adversary A is, to infer whether two nodes $i, j \in V_s$ are connected to each other in G or not. More precisely, whether the link (i, j) between the nodes i and j is missing in G_s, or doesn't exist in G as well.

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 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.

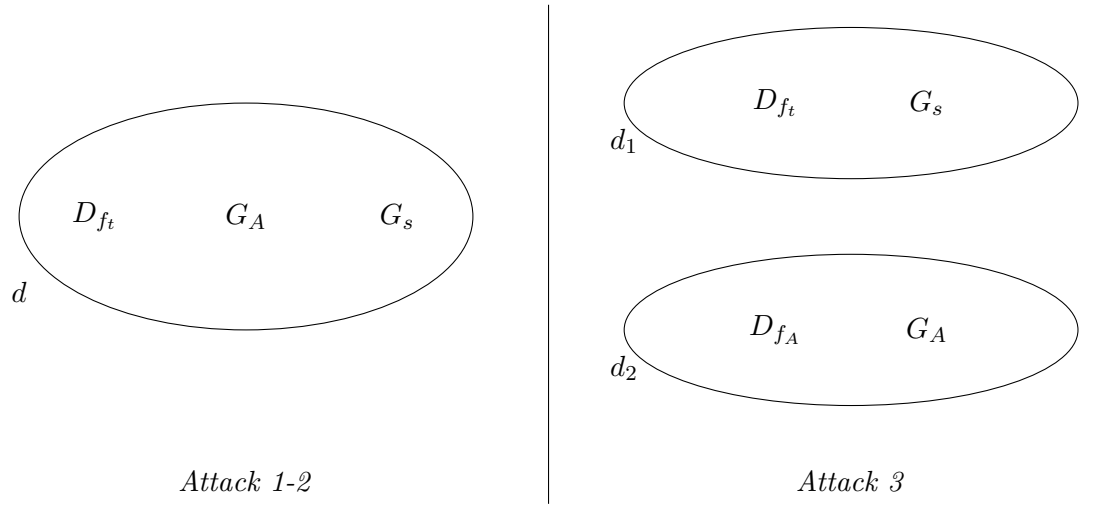

Figure 4.1: Dataset Distributions for Link Stealing Attacks

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 f_t on the subgraph G_A. 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 $u \in G_A$ 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.

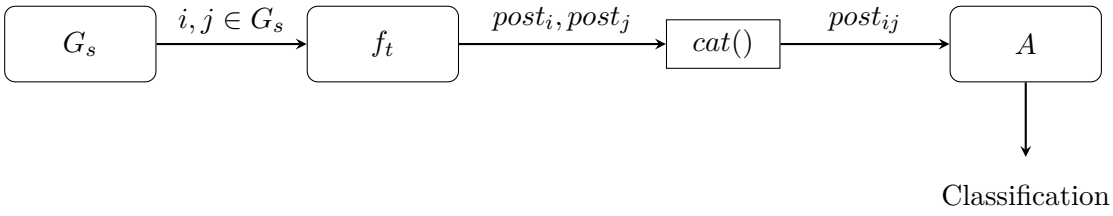

Figure 4.2: Flow Chart - Attack 1

Attack 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 $u \in G_A$ and $v \in G_s$, may 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 in the training graph D_{f_t} and the edge is missing in G_s or not.

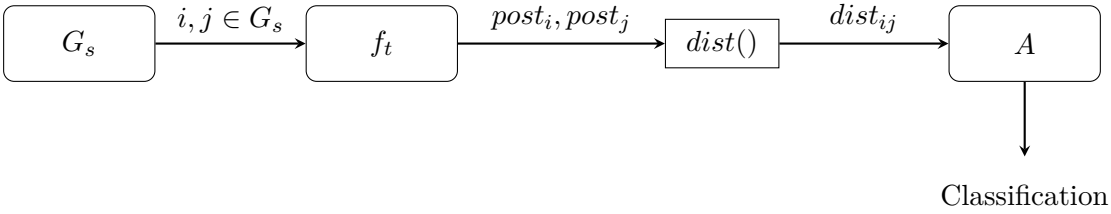

Figure 4.3: Flow Chart - Attack 3

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*.

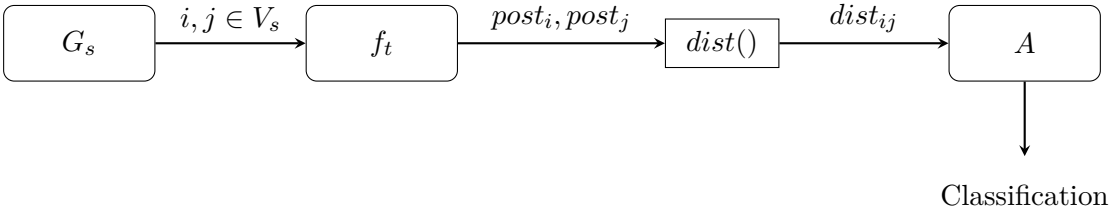

Figure 4.4: Flow Chart - Attack 2

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	$inp_{ij} = cat(post_i, post_j)$	$\|inp_{ij}\| = \|post_i\| + \|post_j\| = 2 * \|post_i\|$
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 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. 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	2708	5429	7	1433
CiteSeer	3327	4732	6	3703
Pubmed	19717	44338	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 attacker 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.

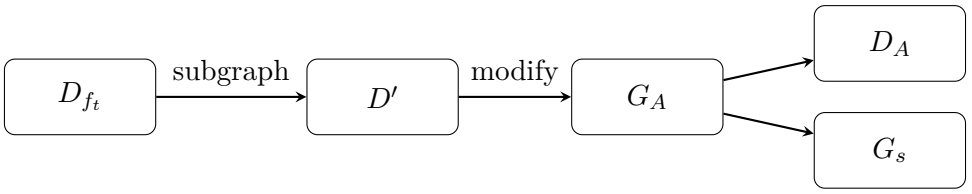

Figure 5.1: Sampling Datasets - Same Dataset Distribution - Attack 1-2

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 $D_A = \{(post_{ij}, l) \mid \forall (i, j, l) \in raw : post_{ij} = concat(f_t(G_A, i), f_t(G_A, j))\}$ for *Attack 1* and $D_A = \{(dist_{ij}, l) \mid \forall (i, j, l) \in raw : dist_{ij} = dist(post_i, post_j)\}$ for *Attack 2*. $f_t(G_A, i)$ returns the node classification output posterior of the target model, when it is queried on i given the adversary's graph G_A. $concat(a, b)$ 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). $dist(a, b) = [Cosine(a, b), ..., Sqeuclidean(a, b)]$, 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 attacker 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.

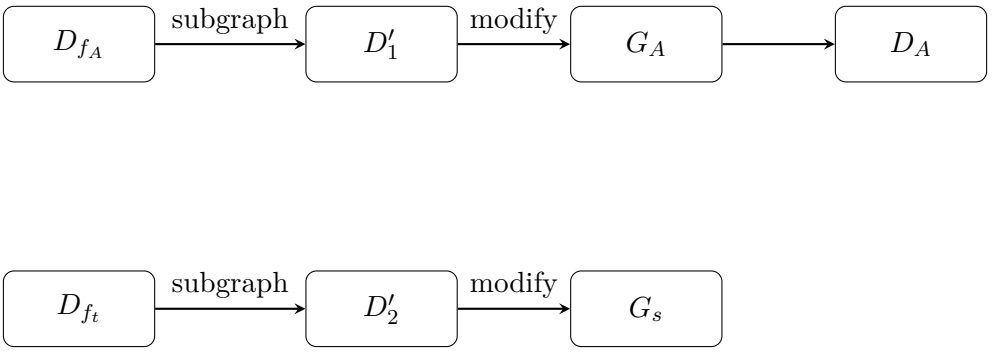

Figure 5.2: Sampling Datasets - Different Dataset Distribution - Attack 3

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 attacker 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. 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 and dropout of 0.5, achieving an average accuracy of 83% for *Cora*, 57% for *CiteSeer* and 89% 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, achieving an average accuracy of 78% for *Cora*, 61% for *CiteSeer* and 89% 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.

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 and dropout of 0.5, achieving an average accuracy of 76% for *Cora*, 57% for *CiteSeer* and 88% 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, 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	83%
	CiteSeer	61%
	Pubmed	89%
GAT	Cora	78%
	CiteSeer	57%
	Pubmed	89%
GCN	Cora	76%
	CiteSeer	57%
	Pubmed	88%

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.

Chapter 6

Evaluation

6.1 Datasets

In total we used 3 different Datasets (Section 5.1), which are common used as 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 provides a good relation between precision and recall.

6.2.1 Precision

A high precision represents a high probability that the prediction of a 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.

6.2.2 Recall

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

6.3 Attack Performance

Attack 1

Attack 2

Attack 3

6.4 Possible Defense

6.5 Summary of Results

Chapter 7

Discussion

7.1 Findings

7.2 Impact

7.3 Future Work

Chapter 8

Conclusion

List of Figures

3.1 Multilayer Perceptron 5
3.2 Undirected Graph 7

4.1 Dataset Distributions for Link Stealing Attacks 12
4.2 Flow Chart - Attack 1 13
4.3 Flow Chart - Attack 3 14
4.4 Flow Chart - Attack 2 14

5.1 Sampling Datasets - Same Dataset Distribution - Attack 1-2 16
5.2 Sampling Datasets - Different Dataset Distribution - Attack 3 17

List of Tables

4.1 Attack Methodology 14

5.1 Dataset Information 15
5.2 Target Model Accuracies 19

8.1 Distance metrics: $f_{t_i}(u)$ represents the i-th component of $f_t(u)$. 31
8.2 Notations 31

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.

[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.