Beyond the Hype: Challenges of Neural Networks as Applied to Social Networks

Anthony Hernandez
University of South Florida

Follow this and additional works at: https://scholarcommons.usf.edu/etd

Part of the Computer Sciences Commons

Scholar Commons Citation
Hernandez, Anthony, "Beyond the Hype: Challenges of Neural Networks as Applied to Social Networks" (2020). Graduate Theses and Dissertations. https://scholarcommons.usf.edu/etd/8218

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Beyond the Hype: Challenges of Neural Networks as Applied to Social Networks

by

Anthony Hernandez

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Adriana Iamnitchi, Ph.D.
Giovanni Luca Ciampaglia, Ph.D.
Lawrence Hall, Ph.D.

Date of Approval:
June 29, 2020

Keywords: Graph Learning, Generative Adversarial Networks, Graph Convolutional Networks, Graph Anonymization, User Activity Forecasting

Copyright © 2020, Anthony Hernandez
Acknowledgments

I would like to express my sincerest gratitude to my advisor Dr. Adriana Iamnitchi for giving me the opportunity to pursue a masters and for her constant support, patience, and feedback. Half of this work wouldn’t exist without Leidos who provided data and DARPA for funding this research work while the other half of the work exists because of NSF for their generous funding. Lastly, I would like to thank all the members of the DSG lab and SocialSim project for making my grad years fun and enjoyable while continuously being fountains of knowledge.
# Table of Contents

List of Tables .................................................................................................................. iii

List of Figures .................................................................................................................. iv

Abstract ............................................................................................................................. v

Chapter 1: Introduction..................................................................................................... 1
  1.1 Objectives ................................................................................................................. 2
  1.2 Contributions ......................................................................................................... 2

Chapter 2: Related Work ................................................................................................. 4
  2.1 Anonymization ....................................................................................................... 4
  2.2 De-Anonymization ................................................................................................. 5
  2.3 Predicting User Activity ......................................................................................... 7

Chapter 3: Methodology ................................................................................................. 8
  3.1 How NetGAN Works .............................................................................................. 8
     3.1.1 Generator and Discriminator of NetGAN ....................................................... 9
     3.1.2 Random Walks ........................................................................................... 10
  3.2 How DCRNN Works .............................................................................................. 11
     3.2.1 Diffusion Convolutional Recurrent Neural Network .................................. 12
     3.2.2 Graph Diffusion Convolution .................................................................... 12

Chapter 4: Datasets ........................................................................................................ 14
  4.1 Graphs for NetGANs ............................................................................................. 14
  4.2 GitHub and Twitter ............................................................................................... 15
     4.2.1 Twitter White Helmets .............................................................................. 16
     4.2.2 Github CVEs ............................................................................................. 17

Chapter 5: Evaluation of NetGAN for Graph Anonymization ........................................ 19
  5.1 Creating Attributed Walks .................................................................................... 20
  5.2 Utility Preservation ............................................................................................... 21
     5.2.1 Sensitivity ...................................................................................................... 22
     5.2.2 Decompression ............................................................................................ 22
     5.2.3 Variance in Synthetic Graphs ...................................................................... 24
     5.2.4 Quality of Synthetic Graphs ....................................................................... 25
  5.3 Anonymization ....................................................................................................... 25
  5.4 Temporal Walks .................................................................................................... 27
Chapter 6: Evaluation of GCN for Social Forecasting

6.1 Problem Definition
6.2 Challenges
6.3 Sparsity
6.4 Granularity

Chapter 7: Conclusion

7.1 Summary on Graph Anonymization
7.2 Summary on Social Forecasting
7.3 Future Work

References
List of Tables

Table 4.1  Simple characteristics of the graphs used to evaluate NetGANs. ............ 16
Table 4.2  Size of the Twitter and GitHub datasets. ..................................... 16
Table 5.1  Classification accuracy across scale-free graphs and timesteps. .............. 21
Table 5.2  Accuracy of generated attributes for the US_air graph. ...................... 21
Table 5.3  Decompression comparison over metric values. ............................... 23
Table 5.4  Graph dissimilarity distance by D-measure.................................... 25
Table 5.5  Quality of graph generators..................................................... 26
Table 5.6  Accuracy of de-anonymization.................................................. 27
Table 6.1  Graph characteristics for each instance of the two networks. ............... 39
Table 6.2  Weekly activity forecasting performance for multiple models. .......... 40
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 5.1</td>
<td>Anonymized versions of the Fb107 graph next to the original.</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Degree distributions of original and synthetic graphs.</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>Sample PCA analyses for the synthetic scale-free graphs.</td>
<td>31</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>A sample instance of temporal walk losses.</td>
<td>32</td>
</tr>
<tr>
<td>Figure 6.1</td>
<td>Weekly activity rates and distribution of inter-arrival times.</td>
<td>37</td>
</tr>
<tr>
<td>Figure 6.2</td>
<td>Sample forecasting plots for models trained on White-Helmets.</td>
<td>40</td>
</tr>
<tr>
<td>Figure 6.3</td>
<td>Sample forecasting plots for models trained on GitHub.</td>
<td>41</td>
</tr>
</tbody>
</table>
Abstract

Recent advances in neural network-based machine learning algorithms promise a revolution in prediction tasks across a variety of domains. Of these, forecasting user activity in social media is particularly relevant for problems such as modeling and predicting information diffusion and designing intervention techniques to mitigate disinformation campaigns. Another potential task is anonymizing social network datasets to facilitate their distribution and promote research. Given the success of deep generative models, it may be possible to use them for anonymization. Social media seems an ideal context for applying neural network techniques, as they provide large data sets and challenging prediction objectives. Yet, our experiments find a number of limitations in the power of deep neural networks and traditional machine learning approaches in predicting user activity on social media platforms as well as creating anonymized networks. Two studies are conducted in this work. The first describes whether a Generative Adversarial Network could produce slightly dissimilar attributed graphs from an original which may implicitly anonymize it. We find issues in how the graph is assembled and how the generator learns attributes for nodes. The second study describes the challenges we encountered while attempting to forecast user activity on two popular social interaction sites: Twitter and GitHub. The custom sequence-to-sequence architecture that is used suffers limitations related to dataset characteristics, specifically temporal aspects of user behavior.
Chapter 1: Introduction

The recent advances in deep neural networks (DNNs) have led to significant improvements in machine learning tasks such as object detection [37, 38] or machine translation [30, 44]. With mounting levels of success in many domains and the huge amounts of graph data available from many areas, it is natural to explore the potential of deep learning when applied to graph data. There have been many successful implementations of deep neural networks on machine learning tasks in the space of networked data, and several comprehensive surveys [47, 52] have been published coalescing all the recent advancements of deep learning in this domain, now dubbed graph learning.

In graph learning, deep learning models are built to perform node or edge classification where a model is trained to classify nodes or edges with missing labels based on known labels in the graph. In link prediction [25], models are trained to estimate the probability of a new edge forming in a graph. Much of the research in this field is in embedding techniques, which are methods to represent the complex nature of graphs into low-dimensional spaces like vectors, normally as a first step in a pipeline to achieve the other tasks mentioned [15, 35].

Less studied in graph learning are temporal-based tasks, i.e., tasks that can process and predict data over time [20]. Given the success of deep learning on datasets represented as networks and the ample volume of data from social media—an ideal candidate for network models—it seems intuitive to apply deep learning to temporal problems in social media, such as forecasting user activity or anonymizing dynamic networks. Although forecasting approaches exist in the literature, they have not been applied to social media networks [26, 50] and to the best of our knowledge the work on anonymizing dynamic graphs is limited and does not include any work involving DNNs.
1.1 Objectives

The main objective of this thesis is to explore the capabilities of neural networks when used on social media networks. To this end, we investigate two problems.

First, whether a generative adversarial network (GAN) can effectively generate an anonymized dynamic social network graph. Protecting private data is a major concern when releasing public datasets. Care has to be taken in maintaining a balance between protection of sensitive information while still retaining any meaningful information for study by others. An ideal graph anonymization technique reduces vulnerability of node identification while maximizing utility, which is defined as a subset of the original graph properties.

Second, whether a Graph Convolutional Network (GCN) can effectively forecast user activities in a social network. Forecasting user activity on social media sites has applications to diffusion processes which occur across a social network. Understanding how software security vulnerabilities spread among an online community of developers, while also predicting the corresponding actions that such developers take across social media, could help devise stronger security practices and develop novel techniques for early detection of vulnerabilities. Similarly, forecasting the effect of (dis)information campaigns by predicting the online activity of the users affected could help develop intervention strategies capable of limiting the impact of these campaigns.

1.2 Contributions

This thesis describes the exploration of two problems, anonymization and forecasting, as applied to social media networks through deep learning approaches.

For anonymization, we experiment with several real-world network datasets and several artificial networks to test the capabilities of a GAN architecture. We present findings showing that the generative architecture used struggles to generate graphs similar to the original in terms of topology and fails to generate a wide variety of synthetic examples. Fur-
thermore, we show possible inherent difficulties in using a random-walk based approach for graph generation.

For Forecasting users’ future activity, we experimented with different time granularity aggregations, and methods of grouping users to investigate whether a higher-level data abstraction could improve predictions but found that accuracy degrades consistently through each aggregation method. We again present findings that show forecasting to be challenging due to the sparse but bursty nature of most users’ time series data as our models tend to severely underpredict values. However, the deep learning approach achieves the best performance against several baseline regression models.
Chapter 2: Related Work

In this section we review literature from several different fields, predicting useractivities on online platforms, graph anonymization, and graph de-anonymization.

2.1 Anonymization

To date, there is still a lack of an effective anonymization technique that can de-
fend against all or even several of the popular structural-based de-anonymization attacks (SDAs) [21]. The challenge of balancing between utility and vulnerability has proven to be difficult, but nevertheless, a large set of techniques have been proposed to address SDAs. These techniques have come to be classified into several categories: Personally Identifiable Trait (PIT) removal, edge perturbation, k-anonymity, and clustering.

Initial methods for anonymizing user-networks were to remove personally identifiable attributes of nodes. This method has been shown to be insufficient to anonymize individual identities in a graph [27, 2]. These early anonymization techniques worked by merely removing identifiable attributes of nodes, such as names, while leaving graph structure intact.

The introduction of structure-based de-anonymization attacks required techniques that work to alter the topology of graphs before release. The most common of these perturbation schemes is to add or delete edges. Specifically, deleting a fraction of edges at random [33], deleting edges at random then creating edges at random while preserving the original number of edges [17], or swapping the incident nodes of two edges to preserve node degree and edge count [48].

Since the introduction of k-anonymity in relational databases [45], techniques have been introduced to modify graphs to fit the properties of being k-anonymous. Pertaining to
graphs, a graph $G(V,E)$ is said to be k-anonymous when for every node $v \in G$ there exists at least $k - 1$ other nodes that are indistinguishable from it. An early example of k-anonymity is in terms of node degree [27]. Over time the technique expanded to include k-anonymity for other graph properties beyond node degree. For example, in [51], k-anonymity is defined over 1-hop and 2-hop neighborhoods.

In the class of clustering anonymization, the main idea is to group nodes into clusters. In [7], the authors proposed to partition nodes into classes and then display the number of interactions between classes. This, however, comes with a greater sacrifice in utility as now the knowledge of which nodes connect to which is lost and replaced with aggregate counts of connections between clusters. A similar approach is taken in [16] where again, nodes are partitioned to maximize utility and then the densities of edges between and within the partitions are released.

More recently, an approach was proposed for anonymization through machine learning by Maag and Denoyer [31]. In this work, a machine learning algorithm automatically selects an optimal anonymization function for a given set of attacks. The algorithm is trained on samples of graphs generated by the different candidate functions and the one whose output reduces the risk for de-anonymization to a minimum in conjunction with utility being preserved is selected. To the best of our knowledge, there doesn’t seem to be established work in anonymizing graphs with DNNs nor ML approaches where the proposed model directly affects the source graph rather than serving as a function selector.

2.2 De-Anonymization

In SDA, there exists two approaches to deanonymizing graphs: seed-based and seedless. In seed-based SDAs, an initial group of nodes with their identities known are used to reveal the remaining users in an anonymized graph. One of the earliest seed-based SDAs was a 2 stage approach proposed in [3], where the authors proposed a ”Sybil” attack before the release of an anonymized dataset. In the first stage, fake users would be created and
form connections with real users in the network. In the second stage, after release of the dataset, an attacker would identify the fake users and proceed to de-anonymize by propagating through the neighborhoods of identified nodes. This approach however, suffers scalability issues as the Sybil users need to be efficiently identified within the released graph as well as needing to be placed in the right area of the network.

Another SDA approach was proposed by Narayanan and Shmatikov [33]. The proposed attack worked in two stages with the use of an auxiliary graph. Initially, a seed set of users which are present in both the anonymized graph and auxiliary graph are identified. These users then have their corresponding nodes mapped between graphs. From the set of seed node pairs, a main propagation stage begins in which the seed mapping is extended to new nodes using the topology of the networks alone. Following the same two-phase approach, Srivasta and Hicks [42] propose three separate models. The first stage identifies a set of seed nodes while the second stage aims to extend the mapping through one of three heuristics: Distance Vectors, Randomized Spanning Tress, and Recursive Subgraph matching.

Seedless approaches, as the name implies, can perform an SDA without relying on an initial set of nodes. Such a seedless approach is proposed in [34]. In this work, a Bayesian method for graph matching is performed by treating an auxiliary graph and anonymized graph as independent samples originating from a fixed original graph. A probability for a correct mapping between node pairs is then derived based on degree and distances to other nodes as evidence.

More closely related to the work presented in this thesis are machine-learning based attacks [40, 19]. In such an attack, a model is trained to classify node pairs between an anonymized graph, $G_A$, and supplementary graph, $G_S$. Both $G_A$ and $G_S$ are subgraphs of $G$, some original graph. The classifier trains on examples of node pairs $(n_a, n_s)$ where $n_a \in G_A$ and $n_s \in G_S$ and classifies the pair as identical when the node pair is actually the same node originating from $G$ and non-identical otherwise. It may be that an anonymization technique that is also based in machine learning may fare better. This becomes the main mo-
tivation behind using GANs to anonymize a social network. A generative method rather than perturbation based could produce better anonymized graphs while still preserving utility.

2.3 Predicting User Activity

Predicting user activity on social networks can encompass a broad range of techniques from classification of activity levels to regression of the volume of activity. In [53], the authors used a modified logistical regression model that incorporates elements of graph structure, personalizing user behavior and time decay. When given a historical set of user activity, the model will classify whether a user will be active or non-active in the near future. There also exists works that parallel our objectives, time series prediction with deep learning, but in different contexts. One such example is [24] where the authors used an LSTM model to predict expected Uber trips during special events like holidays.

In our work, we use a model that leverages the structure of graphs through a GCN along with temporal features. Such models are dubbed Spatio Temporal Graph Neural Networks (STGNNs) [47]. STGNNs have been used in a variety of tasks but the most closely related to this work is traffic forecasting [26, 50]. In both of these works, a model is proposed for the task of time series prediction of traffic sensor networks. A model takes in as features: historical time series of traffic sensors and the underlying road network represented as a graph. The output is a series of predictions for every sensor of what their future traffic volume levels will be. These models have the added advantage of utilizing graph structure in their prediction task, an aspect that could prove beneficial in attempting to forecast user activity on social networks. For example, a user on Twitter who retweeted someone he follows is more likely to retweet that user again in the near future. Keeping track of this influence could improve performance.
Chapter 3: Methodology

This chapter provides details on the two methodologies used for both studies. In this work, we repurpose two existing DNN models for anonymization and social forecasting. For anonymization we used NetGAN [9] which is a GAN originally designed to generate graphs from the same "family" of an original graph. We chose this model over other graph generators because of its ability to handle larger graphs and because its random-walk approach was purported to successfully capture many known properties of a source graph while still producing significant variance between its outputs.

For social forecasting, we used the Diffusion Convolutional Neural Network (DCRNN) which was originally used for traffic forecasting [26]. The model can take in historical time series and a corresponding graph and outputs future time series. This model was chosen for its use of graph convolutionals that operated over weighted and directed edges showing good generalizability as well as handling temporal data. Since social networks can be directed and are almost always dynamic, DCRNN can be a good candidate model for our task.

3.1 How NetGAN Works

A typical GAN has two core components: a generator that learns some distribution and transforms output from a noise function to fit to the learned distribution, and a discriminator that must learn to distinguish between samples from the generator and samples from the real distribution while providing feedback to the generator. Competition drives performance gains for both the discriminator and generator in their respective tasks. The generator continuously improves the quality of its samples and the discriminator continu-
ously improves its detection abilities until eventually synthetic samples are indistinguishable from genuine.

The goal of the generator in NetGANs is to learn a distribution of edges amongst walks and generate synthetic walks that closely resemble the real ones sampled from an original graph. These random walks are created through an algorithm that mixes breadth-first search and depth-first search in order to create paths over nodes in a graph that characterize its structure. After training, a large set of synthetic walks are generated. From these walks a transition matrix is assembled where an entry at \((i, j)\) denotes the number of times the walks contained a transition between nodes \(i\) and \(j\). This transition matrix represents the probability distribution of edges that were present in the synthetic walks. Sampling from this matrix yields a set of edges that represent the final synthetic graph. The sampling process ensures that all nodes are present and contain an edge.

### 3.1.1 Generator and Discriminator of NetGAN

The generator \(G\) consists of an LSTM layer and is modeled as a sequential process with the number of steps, \(T\), equal to the random walk length which is set at the start of training. When timestep \(t = 1\), the generator samples from a latent space \(z\) drawn by a normal distribution to produce \(m_0\), the initial values for the memory state of the LSTM represented as a vector of randomized values. At each time step \(t > 1\) the generator produces a probability distribution for the next node \(p_t\) and a memory state \(m_t\) for the LSTM. The logits \(p_t\) is sampled to produce the node \(v_t\), then \(m_t\) and \(v_t\) are both passed back to the model to create logits \(p_{t+1}\) and \(m_{t+1}\). The logit vector \(p_t\) can have a high dimension as it must be the same size as the number of nodes \(N\) in a graph, yet \(N\) can be upwards of over several thousand causing increased computational overhead. To reduce overhead, the generator does not directly output \(p_t\) but rather a vector denoted as \(o_t \in \mathbb{R}^H\) where \(H \ll N\). The vector \(o_t\) is up-projected to \(p_t\) using a matrix tensor \(W_{up} \in \mathbb{R}^{H \times N}\). After \(v_t\) is sampled, its one-hot vector representation is down-projected with another matrix tensor \(W_{down} \in \mathbb{R}^{N \times H}\). Using
these projection matrices allows better handling for larger graphs by ensuring the LSTM layer can always create hidden states of a smaller fixed size.

A generalized equation for calculating \((p_t, m_t)\) is defined as follows:

\[
(p_t, m_t) = \begin{cases} 
  f_\theta(g_\theta(z), 0), & \text{if } t = 1 \\
  f_\theta(m_{t-1}, v_{t-1}), & \text{otherwise}
\end{cases}
\]  

(3.1)

where \(f_\theta\) is a neural network parameterized by \(\theta\) and \(g_\theta\) is a parametric function for transforming \(z\). Sampling the logits \(p_t\) to determine the next node in a walk can formulated as follows:

\[v_t \sim \text{Cat}(\sigma(p_t))\]  

(3.2)

where \(\text{Cat}\) represents a categorical sampling and \(\sigma(.)\) represents the sigmoid function.

The discriminator also consists of an LSTM layer operating in a sequential process. Given a walk, at every time step \(t\), a one-hot vector denoting \(v_t\) is fed as input. Once the entire walk is fed in, a single final score is the output denoting the likelihood of the walk being real. NetGANs uses the Wasserstein GAN [1] approach for training. In this approach, the cost function used is the earth-mover distance which has been shown to prevent mode-collapse and create more stable training of models. Optimization is performed with Adam [22], a fast but unstable stochastic gradient descent implementation.

3.1.2 Random Walks

In NetGANs, random walks are used as the training examples as well as the output format of the generator. Training examples are generated through a 2\textsuperscript{nd} order walk. To carry out walks there are two parameters, \(p\) and \(q\), that guide the walk by setting the transition probabilities between the node currently being visited \(v\), and the next node to visit, \(x\), as a function \(\alpha_{pq}(t,x)\) of the shortest path distance between the last node visited \(t\) and neighbors.
of $v$ (which includes $t$). Formally, the function is defined as:

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p}, & \text{if } d_{tx} = 0 \\ 1, & \text{if } d_{tx} = 1 \\ \frac{1}{q}, & \text{if } d_{tx} = 2 \end{cases}$$ (3.3)

The first node in any walk is randomly chosen, while subsequent nodes are sampled using the above transition probabilities. Walks generated in this fashion are 1-dimensional vectors of length $L$ where the value of an entry represents the unique identifier (ID) of nodes.

We wanted to generalize the capabilities of NetGAN to generate attributed graphs in order to anonymize them. To do so, some changes to the architecture needed to be made. The random walk vectors are made 2-dimensional by having the walker pick up node attributes along with node ID. The generator is subsequently modified to generate attributes by producing one additional element at each time step along with the logits $p_t$, thus generating $p_t + 1$ sized vectors. This element is sampled to create the final binary attribute for the corresponding sampled node. The node ID and attribute value are concatenated and then appended to the walk being generated to form the final 2D attributed walk vector.

### 3.2 How DCRNN Works

Graph learning introduced a new architecture called Graph Convolutional Networks (GCNs) [23, 14]. This architecture is a generalized form of traditional Convolutional Neural Networks that can operate on non-euclidean data such as graphs. GCNs create hidden features for a node by aggregating the feature value with those of its nearest neighbors. This ability has led to the popularization of GCNs as building blocks in more complex DNN architectures applied to graphs.

In this thesis we use the *Diffusion Convolutional Recurrent Neural Network* (DCRNN) architecture proposed in [26] where it was originally used to forecast traffic metrics over road
networks. This architecture was chosen for the following reasons: First, it contains a GCN allowing us to incorporate graph structure when learning a diffusion process. Second, it handles dynamic data allowing us to use historical activities to predict future ones. Last, it achieved state-of-the-art performance in its original task, thus establishing its effectiveness. These reasons make DCRNN seem a promising choice for our task of forecasting user activities and provided an easy framing of the problem: predicting sequences from sequences.

3.2.1 Diffusion Convolutional Recurrent Neural Network

The Diffusion Convolutional Recurrent Neural Network (DCRNN) is a recurrent neural network that incorporates two well-known neural network layers: Gated Recurrent Units (GRU) [12] and GCNs inside of the sequence to sequence architecture described in [44]. DCRNN is designed to use historical time series to predict future time series. To do so, there is an encoder which takes in the historical time series and outputs a final hidden state. The decoder will use this final state as a seed to predict the next several states of a future time series.

In both encoder and decoder, there is a modified GRU where its original matrix multiplications have been replaced with a graph diffusion convolution described below (Chapter 3.2.2).

The model learns to maximize the likelihood of generating the future time series. In maximizing the likelihood, DCRNN creates hidden features of a historical graph signal which contain spatial and temporal dependencies. When training the model, scheduled sampling is used [6]. This technique introduces a probability to feed the decoder ground truth data as seeds to improve model performance.

3.2.2 Graph Diffusion Convolution

The diffusion convolution operation defined in [26] can capture spatial dependencies between nodes in a graph by assuming there is a diffusion process present in the network.
This diffusion process is represented in truncated form as \((D^{-1}A)^k\), where \(A\) is the adjacency matrix for \(G\), \(D = \text{diag}(A1)\) and 1 is a vector of ones. The parameter \(k\) represents the diffusion step, which corresponds to the number of steps that we let a diffusion process take. In practice, \(k\) is held constant at \(k=2\), as higher degrees do not improve model performance significantly but increases overhead as \(D^{-1}A\) must be raised to \(k\).

We can use this diffusion transition matrix as a convolution over graph signal \(X \in \mathbb{Z}^N\) with filter weights \(f_\theta\). The resulting diffusion convolution operation is thus:

\[
X \star Gf_\theta = \sum_{k=0}^{2}(\theta_k(D^{-1}A)^k)X
\]

where \(\theta \in \mathbb{R}^k\) are the learned parameters for the filter. Effectively, this convolution will aggregate a node signal with its neighbors by using the transition matrix taken out to \(k\) diffusion steps. The resulting hidden features contain neighbor influence on a graph signal.
Chapter 4: Datasets

This chapter provides details about the different datasets used in both studies. We noticed that in the original work for NetGAN, the two datasets used, cora-ml and citeseer, were highly similar in terms of graph metrics such as assortativity. In our experiments a wider variety of graphs were used that were both real-world and synthetic to better explore the capabilities NetGAN.

For testing DCRNN we used two real-world datasets, Github and Twitter, provided as part of the ongoing SocialSim project funded by DARPA. The Github dataset contains a collection of user events that occurred over repos connected to a number of selected Common Vulnerability Exposures (CVEs) while the Twitter dataset contains user activities related to the WhiteHelmets, a volunteer organisation in Syria that has been subject to disinformation campaigns by their government.

4.1 Graphs for NetGANs

In testing attributed walk generation of NetGAN, synthetic graphs and a real world graph were used with a diffusion process applied to them in order to create attributes. An R software package called NetDiffuseR [46] was used to run the diffusion process. This package was chosen as it contains a suite of algorithms for analysis, visualization, and simulation of diffusion processes. We used one such diffusion process provided in this package called adoption of innovation. This diffusion is represented as a binary attribute where 1 refers to adopted behavior and 0 as the starting behavior. A node within this model adopts the innovation when their exposure surpasses a randomized threshold value and the threshold
is random for every node. Exposure for a node is represented by the number of neighbors that have already adopted the behavior. The simulation consists of the following steps:

1. A scalefree graph of 1000 nodes is created following the preferential attachment model [5].

2. 30% of the nodes are randomly selected to be initial adopters, with a time of adoption (TOA) set to 1.

3. Randomize the threshold levels of every node in the graph.

4. Simulation begins at $t = 2$. In each time step $t$, if a node’s exposure surpasses its threshold, label it as adopted and set $TOA = t$. Stop when a timestep has no adopters.

Synthetic graphs were used as they allowed finer control over the diffusion processes that yielded the attributes for nodes. Graphs were chosen when they yielded a diffusion process with a steady growth of adoption that culminated in a majority of nodes adopting the innovation after several time steps. Scale-free [5] synthetic graphs were used as they more closely resemble social networks. The real world graph used was US_airports which models the major airports across the United States. This graph was chosen using the same preferences for the synthetic graphs.

Experiments were conducted to measure how well NetGANs preserves the qualities of an original graph as well as how varied synthetic graphs were in terms of metrics. Previous work evaluated the performance of traditional graph generators in preserving the utility of networks [19]. We expanded upon these evaluations by adding NetGANs to the comparisons. We examine two real-world graphs used in the evaluation, Fb107 representing social circles from Facebook and Ca-GrQc which represents a collaboration network from arXiv about general relativity papers. Characteristics for these graphs are shown in Table 4.1.

### 4.2 GitHub and Twitter

For forecasting we use two datasets that are different in terms of the context they represent and the social media platform from which they are collected. The first dataset
Table 4.1: Simple characteristics of the graphs used to evaluate NetGANs. In order, the metrics are number of nodes, number of edges, clustering coefficient, and assortativity.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Nodes</th>
<th>Edges</th>
<th>cc</th>
<th>assort</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>3000</td>
<td>44700</td>
<td>0.01</td>
<td>0.003</td>
</tr>
<tr>
<td>G2</td>
<td>3000</td>
<td>89890</td>
<td>0.02</td>
<td>0.003</td>
</tr>
<tr>
<td>US_air</td>
<td>1572</td>
<td>17215</td>
<td>0.42</td>
<td>-0.118</td>
</tr>
<tr>
<td>Fb107</td>
<td>1034</td>
<td>26749</td>
<td>0.52</td>
<td>0.43</td>
</tr>
<tr>
<td>Ca-GrQc</td>
<td>5242</td>
<td>14496</td>
<td>0.56</td>
<td>0.66</td>
</tr>
</tbody>
</table>

captures Twitter posts related to the Syrian volunteer organization "White Helmets". The second dataset is a collection of user events related to common vulnerability exposure (CVE) identifiers in GitHub, a collaborative software platform. These datasets were provided as part of the on-going SocialSim project supported by the Defense Advanced Research Projects Agency (DARPA). Table 4.2 shows the basic characteristics of the SocialSim datasets used in this study.

Overall, we had over 1 million Twitter posts done by 222,485 users. Of these, tweets represent 10%, retweets 76%, quotes 5% and replies 6%. For Twitter, we only focus on the most common activity, retweeting, with 802,872 retweets done by 180,961 users in our dataset. In GitHub, we had 745,169 events done by 30,525 users on 30,133 software repositories, of which 58% were Push events, 20% Issue events, 15% Pull Request events, and the rest under 3% each. We focus only on Push events as they are the majority in GitHub, with 431,560 Push events from 14,780 users on 22,334 software repositories.

Table 4.2: Size of the Twitter and GitHub datasets.

<table>
<thead>
<tr>
<th>Dataset (Event)</th>
<th>Activities</th>
<th>Users</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter (Retweets)</td>
<td>802,872</td>
<td>180,961</td>
<td>04/01/18–04/30/19</td>
</tr>
<tr>
<td>GitHub (Pushes)</td>
<td>431,560</td>
<td>14,780</td>
<td>01/01/15–03/31/18</td>
</tr>
</tbody>
</table>

4.2.1 Twitter White Helmets

The White Helmets are a search-and-rescue volunteer organization operating in Syria. During these operations, members of the White Helmets document bombings, chemical at-
tacks, and potential war crimes committed by the Assad regime [41]. These actions have provoked the government to mount orchestrated disinformation campaigns on social media to undermine the credibility of White Helmets [43].

This dataset was collected from Twitter over 13 months starting in April 2018 using GNIP, a data collection API tool. Twitter is a popular micro-blogging platform focused on broadcasting messages (tweets) and can be used for information and disinformation campaigns [10, 13] run by industry, state-sponsored organizations, etc. Users can interact with such messages via retweets, replies or quotes. Tweets can be tagged with hashtags, which are used to describe a topic of interest and allow tweet categorization. The Twitter stream was set to pull tweets with messages and hashtags containing any mention of White Helmets in multiple languages, including English, French, Arabic, Spanish, German and Russian.

Each record (retweet) in the processed dataset consisted of the following fields: the unique identifier of the retweet, the user identifier of the author of the retweet, the retweet timestamp, and the original tweet ID and its author ID. We constructed the corresponding retweet network, a static directed graph where users are connected to each other through retweeting one another. A user has an outgoing edge to a neighbor if that neighbor retweeted the user within the time frame of the dataset. The resulting network included a large number of small connected components. Thus, we decided to focus only on the largest strongly connected component of the network for our experiments. This sub-graph consists of 6,376 users, who performed 266,975 total activities collectively. Our choice for reducing the size of the dataset by focusing on a subset of the data was motivated by the scalability limitations of the graph convolutional layer.

4.2.2 Github CVEs

GitHub is an open-source collaboration platform where contributions to software repositories take place via events such as commits, pull-requests, pushes and issues. Public repositories can be forked (i.e., copied) to make use of the available code and additionally
make modifications. Users also have the option to watch repositories to receive alerts on updates, such as pull-requests or new issues being raised.

This dataset contains information about CVE exploits posted in public GitHub repositories related to cyber-security software vulnerabilities. For data collection, a subset of CVE identifiers recorded with the National Vulnerability Database (NVD) between January 2015 to March 2018 was selected. These CVE identifiers are related to those exploited by Advanced Persistent Threats (APTs), which consists of state-sponsored groups that aim to gain unauthorized access to a network to steal sensitive information. From the GitHub public dataset only those software repositories that contained any mention of at least one CVE identifier from our subset in their description were selected. Our dataset contains all software repositories with at least one mention of a CVE of interest in any of the comments associated with push events.

We analyzed a dataset containing the following information: the unique identifier of the software repository, the unique identifier of the user who performed the push event, the exact timestamp when the push event occurred, and the CVE identifier related to the event. Each CVE that a software repository is tagged with is also associated with one or more products (e.g., Windows 10 or Outlook) and its respective vendor (e.g., Microsoft).

We model this dataset as an undirected graph as follows: nodes are represented by software repositories, and repositories are connected to each other if they are targeted by the same CVE identifier. An edge exists between two repositories only if both contain GitHub events pertaining to the same CVE identifier. The largest connected component for our experiments was chosen so isolated repositories were ignored. The component contained 99% of the repositories in the dataset. The sub-graph contained 22,052 software repositories associated with 416,414 push events done by 13,116 users, and related to 1,391 products and 498 vendors.
Chapter 5: Evaluation of NetGAN for Graph Anonymization

In this chapter we explore one of the approaches for generating graphs with deep learning. Our motivation for generating graphs was to test whether generating graphs with particular characteristics, as learned from real graph datasets, can be a feasible approach to anonymizing graphs. Since NetGANs supposedly generates graphs within the same family of some original graph, then all synthetic creations should closely resemble the original, but ideally remove identifiable structural aspects of the original topology. Our initial goal was to anonymize in this way graphs with node attributes and maybe even dynamic graphs.

We tested this idea by first generalizing NetGAN to incorporate attributes in its synthetic walk creation. Initial experiments show that the generator struggles to produce nodes with attributes. As a result, we reverted back to the original architecture and tested NetGAN’s ability to generate graphs over a series of real-world networks to see if at least utility can be preserved on the original architecture. The experiments show that NetGAN is actually not suitable for working with real-world graphs because of inherent issues within the random-walk process and from how a graph is assembled from the generator output, which results in poor variance between examples and poor utility of synthetic graphs. Because of the well documented tension between graph utility and anonymity [21, 33], we wanted to confirm that the anonymity of the synthetic graphs (with low utility, that is, low resemblance to the original graph) is high. Thus, we evaluated NetGAN’s ability to anonymize graphs by measuring the resistance of the output graphs to a well known seed-based SDA [33] and compared this resistance to classic anonymization methods of k-degree [28] and edge swapping [48]. As expected, NetGAN has the highest resistance to the SDA as a result of how greatly the synthetic examples deviate from the original graph.
5.1 Creating Attributed Walks

The discriminator was recreated on its own using the same architecture outlined in [9]. Specifically, the architecture consists of a single Long Short-Term Memory (LSTM) layer followed by a Fully Connected (FC) layer with sigmoid activation, trained using the Adam optimizer. Instead of scoring walks, however, the task became binary classification with binary-cross entropy as the loss function. The discriminator is tasked with merely classifying whether a walk is synthetic. In the original model, the prescribed number of epochs was 10,000 however, in training the classifier over the synthetic graphs, epochs over 1,000 yielded sufficient accuracy. For a given timestep, a set of real and synthetic walks were generated using the random walk process described in Chapter 3 with the exception that synthetic walks had their attributes replaced with randomly chosen values. Table 5.1 shows results of the experiments averaged over 10 trials for each timestep. From the results it seems that the discriminator is capable of correctly identifying attributed walks. It also seems that models tend to perform worse as the number of adopted nodes increases as is evidenced by poorer accuracy at later time steps in the diffusion.

The generator was next tested in how well it can produce attributed walks by using the modified architecture described in Chapter 3. We trained a model on sampled timesteps of the US_airport graph and generated sets of synthetic attributed walks. As the generator is stochastic, the attribute value for the same node ID can change when observed between different walk sequences. To determine the final synthetic attribute value for a node’s ID, the occurrences for which the attribute value is 1 or 0 are tallied and then the majority class is selected. After the class is determined, we compute the F1 scores between the generated classes and the true classes. The final results can be seen in Table 5.2 where it is apparent that the model struggles to generate closely matching nodes attributes. While it is not the goal to match these adopting nodes exactly, it would be ideal to have only minor errors so as to preserve the utility of the original graph as best as possible. The only model that could produce highly similar infected nodes is in the trivial case of timestep 23 where 95%
Table 5.1: Classification accuracy across scale-free graphs and timesteps.

<table>
<thead>
<tr>
<th>Graph</th>
<th>TimeStep</th>
<th>fraction adopted</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>1</td>
<td>0.3</td>
<td>0.78</td>
</tr>
<tr>
<td>G1</td>
<td>4</td>
<td>0.59</td>
<td>0.79</td>
</tr>
<tr>
<td>G1</td>
<td>6</td>
<td>0.68</td>
<td>0.73</td>
</tr>
<tr>
<td>G1</td>
<td>8</td>
<td>0.76</td>
<td>0.61</td>
</tr>
<tr>
<td>G2</td>
<td>1</td>
<td>0.3</td>
<td>0.79</td>
</tr>
<tr>
<td>G2</td>
<td>6</td>
<td>0.63</td>
<td>0.77</td>
</tr>
<tr>
<td>G2</td>
<td>20</td>
<td>0.84</td>
<td>0.76</td>
</tr>
</tbody>
</table>

Table 5.2: Accuracy of generated attributes for the US_air graph. Several time steps were chosen to train a model over.

<table>
<thead>
<tr>
<th>Timestep</th>
<th>actual infected (%)</th>
<th>predicted infected (%)</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.4</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>12.3</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>14</td>
<td>41.4</td>
<td>15.5</td>
<td>0.233</td>
</tr>
<tr>
<td>23</td>
<td>95.4</td>
<td>1.0</td>
<td>0.977</td>
</tr>
</tbody>
</table>

of the nodes are infected resulting in a generator learning to merely choose the majority class. Given the generator struggles to produce attributed walks, the scope was reduced to examine if this architecture could create anonymized graphs without attributes.

5.2 Utility Preservation

As mentioned earlier, anonymization of a graph comes at the cost of utility.

The following section outlines several experiments where each is motivated by one of the following questions:

- What effect do hyper parameters have on the output of the graph?
- Does the use of Wdown and Wup, the (de)compression matrices used to reduce overhead, adversely affect the quality of synthetic graphs?
- How much variance is there in the synthetic graphs?
- How well does NetGan perform compared to popular graph generative methods?
In answering these questions we discovered two major shortcomings of the model that the original paper never mentioned: (i) the generator exhibits a lack of variance, and (ii) it loses utility of generated samples.

5.2.1 Sensitivity

In the original NetGAN paper [9], it is stated that most hyper-parameter values are not critical for performance, aside from random walk length, and that their values need only be in an acceptable range. To verify this, the US_airport graph was used as the dataset and a model was trained using the default values provided in the original work. We then tweaked the value of one parameter at a time while holding all others at their default. For each model trained, a graph was generated and metrics recorded.

This experiment differs from the sensitivity analysis done in the original work by not focusing on the link prediction performance of the models and instead on the generation performance in terms of metrics. The parameters we tested included: discriminator iterations which was how many times the discriminator updated per 1 update of the generator, the length of randomwalks, the batchsize, both p and q which are used for creating the real random walks, the learning rate, the number of hidden units in both generator and discriminator, as well as the sizes for the projection matrices $W_{down}$ and $W_{up}$. Our analysis confirms that parameters’ values do not significantly affect the final outcome in terms of graph metrics and that training is actually highly stable when observing the losses in conjunction with metric values.

5.2.2 Decompression

Part of the architecture in NetGANs is the use of projection matrices which allows the output size of the generator to be far less than the number of nodes in a graph, thus reducing computational overhead. Generating nodes in a much lower dimension than their original could lead to information loss. We believe that the potential information loss that occurs
Table 5.3: Decompression comparison over metric values. In order, the metrics are average degree of nodes, the number of triangles, assortativity, clustering coefficient, and edge density (fraction of existing edges over total possible).

<table>
<thead>
<tr>
<th>Decompression</th>
<th>avg.deg</th>
<th>triangles</th>
<th>assort</th>
<th>cc</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>original</td>
<td>18.61</td>
<td>143999.0</td>
<td>-0.120</td>
<td>0.41</td>
<td>8.0</td>
</tr>
<tr>
<td>none</td>
<td>21.9</td>
<td>94487.0</td>
<td>-0.096</td>
<td>0.23</td>
<td>5.0</td>
</tr>
<tr>
<td>partial</td>
<td>21.9</td>
<td>88615.0</td>
<td>-0.094</td>
<td>0.22</td>
<td>6.0</td>
</tr>
<tr>
<td>full</td>
<td>21.9</td>
<td>8249.0</td>
<td>0.011</td>
<td>0.038</td>
<td>4.0</td>
</tr>
</tbody>
</table>

due to encoding a large dimension to a small one, such as a vector of size 1,000 to a size of 128, worsens model performance. We studied these matrices as part of experimenting with NetGAN, but ultimately found them to benefit the generator in its learning task. To test for information loss, two experiments were conducted, one which removed only $W_{down}$ and one that removed both $W_{down}$ and $W_{up}$.

The reason for two experiments is that when $W_{up}$ is removed, the LSTM representing the Generator must have its cell size increased to $N$, the number of nodes in a graph, leading to a blow-up in computational overhead. Having an architecture where the input to the next step is high dimensional, but the output of the current step remains low dimensional could be a potential compromise if it is seen that there is information loss in the original architecture. In both experiments, we took the default parameters prescribed by the original paper, then trained a model on the US_airport graph using partial decompression, total decompression, and the original architecture to see if there was any difference in the outputs of the graph in terms of standard metrics including: average degree, number of triangles, gini coefficient, assortativity, clustering coefficient, and edge density.

Table 5.3 shows that there is no significant change in quality of the synthetic graphs between partial and original architecture, but total decompression actually performs worse, perhaps due to the increased number of units required to be trained.
5.2.3 Variance in Synthetic Graphs

Having quality examples from a generator is only half the challenge. Ideally, a generator should also be able to produce a significant variety of examples. To test NetGANs variance, we use the metric proposed in [39] called D-measure. This metric returns a value from 0 to 1, is shown to identify and quantify topological differences in graphs, and return a value of 0 for isomorphic graphs. We train a model on two different graphs, fb107 and CaGrQc, using the default parameters prescribed, and then 10 synthetic graphs were created from each model. The D-measure was calculated between each graph in the synthetic set as well as to their original.

This may not be a sign of mode collapse in the generator, that is, generated samples vary little from one another, but rather an unavoidable consequence of the graph assembly in NetGANs. The generator can produce a wide variety of random walks but aggregating the walks will typically yield the same graph. To assemble the graphs in NetGAN, walks are sampled from the generator which in turn are transformed into a transition matrix of size $N \times N$ where entries denote the probability of an edge existing amongst the walks. A number of edges are subsequently sampled from this transition without replacement until the count equals the original number of edges. The resulting set of sampled edges becomes the new synthetic graph. In the original paper, a large set of walks are sampled from the generator, 500,000 walks of length 16. Naturally, counting transitions over this many walks will converge to some stable probability distribution despite using different sampled sets. Similarly, sampling a large amount of edges from this probability matrix, the number of edges from an original graph, should yield almost the same graph every time. The original paper never discusses these potential issues and no explicit mention is made of the generator only being able to produce a single synthetic version of an original graph.
Table 5.4: Graph dissimilarity distance by D-measure. Distance to original refers to the average distance of each generated graph in a set to the original while inter-distance refers to the average distance between elements of the generated set.

<table>
<thead>
<tr>
<th>Graph</th>
<th>distance to original</th>
<th>inter-distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fb107</td>
<td>0.42 ± 0</td>
<td>0.0022 ± 0.0011</td>
</tr>
<tr>
<td>Ca-GrQcc</td>
<td>0.48 ± 0</td>
<td>0.006 ± 0.002</td>
</tr>
</tbody>
</table>

5.2.4 Quality of Synthetic Graphs

We finally experiment to see how NetGANs compares to other popular generative methods for graphs. In another work which tested utility preservation [19], 3 styles of generators are used to produce synthetic versions of the four graphs. Metrics were calculated on the synthetics graphs and compared. In the original work, 100 trials were created for each method but having seen how stable NetGANs is in training and how little variance a model has, our addition to the comparison is over 10 trials. Table 5.5 shows that NetGans struggles on preserving assortativity and fraction of degree-1 nodes but can match the other metrics reasonably well. Data shows that Dk2 generators can preserve traits of the original far better than NetGAN across every graph and metric.

5.3 Anonymization

We conducted experiments to measure the level of anonymization that NetGAN can naturally provide. For these experiments we again used Fb107 and Ca-GrQcc as the graphs to be anonymized. In addition, we used SecGraph [21], a publicly available software package, to perform the anonymization and de-anonymization.

To evaluate the effectiveness of NetGANs, we compared its resistance to an SDA with two other popular anonymization methods: k-degree anonymity and edgeswapping. The SDA chosen for these graphs was the Narayanan-Shmatikov [33], one of the first seed-based attacks. To select the seeds for a given graph, the first 50 nodes that appeared across a list of the largest cliques in the anonymized version were chosen in order to give the SDA
Table 5.5: Quality of graph generators. In order, the metrics are number of nodes $N$, number of edges $E$, edge density $d$ (fraction of existing edges over total possible edges), assortativity, transitivity, average path length $L$, and percent of degree-one nodes deg-1.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Generator</th>
<th>N</th>
<th>E</th>
<th>d</th>
<th>assort</th>
<th>trans</th>
<th>L</th>
<th>deg-1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>original</td>
<td>1034</td>
<td>26749</td>
<td>0.0500</td>
<td>0.4316</td>
<td>0.5045</td>
<td>2.9517</td>
<td>1.45</td>
</tr>
<tr>
<td>Fb107</td>
<td>0K</td>
<td>1034</td>
<td>26749</td>
<td>0.0501</td>
<td>-0.0029</td>
<td>0.0501</td>
<td>2.0210</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>1K</td>
<td>1034</td>
<td>26749</td>
<td>0.0501</td>
<td>-0.0961</td>
<td>0.1466</td>
<td>2.1965</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td>2K</td>
<td>1034</td>
<td>26749</td>
<td>0.0501</td>
<td>0.4316</td>
<td>0.3161</td>
<td>2.4020</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td>ERGM-apl</td>
<td>1034</td>
<td>26749</td>
<td>0.0501</td>
<td>0.0017</td>
<td>0.0504</td>
<td>2.0193</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>ERGM-cc</td>
<td>1034</td>
<td>26749</td>
<td>0.0501</td>
<td>0.4293</td>
<td>0.5038</td>
<td>2.8796</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>ERGM-dc</td>
<td>1034</td>
<td>26749</td>
<td>0.0501</td>
<td>0.3747</td>
<td>0.1627</td>
<td>2.1197</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>LF (m=2)</td>
<td>1034</td>
<td>2066</td>
<td>0.0039</td>
<td>0.1425</td>
<td>0.2173</td>
<td>10.2155</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>LF (m=5)</td>
<td>1034</td>
<td>5165</td>
<td>0.0097</td>
<td>0.2308</td>
<td>0.2463</td>
<td>5.5336</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>LF (m=10)</td>
<td>1034</td>
<td>10330</td>
<td>0.0193</td>
<td>0.2733</td>
<td>0.2164</td>
<td>3.6806</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>NetGAN</td>
<td>1034</td>
<td>26749</td>
<td>0.0500</td>
<td>0</td>
<td>0.08</td>
<td>2.12</td>
<td>0.02</td>
</tr>
<tr>
<td>Ca-GrQc</td>
<td>original</td>
<td>5242</td>
<td>14496</td>
<td>0.0011</td>
<td>0.6592</td>
<td>0.6298</td>
<td>3.8047</td>
<td>22.83</td>
</tr>
<tr>
<td></td>
<td>0K</td>
<td>5242</td>
<td>14496</td>
<td>0.0011</td>
<td>-0.0011</td>
<td>0.001</td>
<td>5.2155</td>
<td>2.22</td>
</tr>
<tr>
<td></td>
<td>1K</td>
<td>5241</td>
<td>14484</td>
<td>0.0011</td>
<td>-0.0355</td>
<td>0.0077</td>
<td>4.0002</td>
<td>22.83</td>
</tr>
<tr>
<td></td>
<td>2K</td>
<td>5241</td>
<td>14484</td>
<td>0.0011</td>
<td>0.6593</td>
<td>0.271</td>
<td>1.041</td>
<td>22.83</td>
</tr>
<tr>
<td></td>
<td>ERGM-apl</td>
<td>5241</td>
<td>14484</td>
<td>0.0011</td>
<td>0.039</td>
<td>0.0064</td>
<td>5.439</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>ERGM-cc</td>
<td>4507</td>
<td>14484</td>
<td>0.0014</td>
<td>0.6804</td>
<td>0.6278</td>
<td>5.6361</td>
<td>10.43</td>
</tr>
<tr>
<td></td>
<td>ERGM-dc</td>
<td>5237</td>
<td>14484</td>
<td>0.0011</td>
<td>0.4547</td>
<td>0.079</td>
<td>5.5294</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>LF (m=2)</td>
<td>5242</td>
<td>10482</td>
<td>0.0008</td>
<td>0.1536</td>
<td>0.2132</td>
<td>13.0612</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>LF (m=5)</td>
<td>5242</td>
<td>26205</td>
<td>0.0019</td>
<td>0.24</td>
<td>0.2348</td>
<td>7.1527</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>LF (m=10)</td>
<td>5242</td>
<td>52410</td>
<td>0.0038</td>
<td>0.2771</td>
<td>0.1895</td>
<td>4.7513</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>NetGAN</td>
<td>4158</td>
<td>13422</td>
<td>0.0016</td>
<td>-0.0027</td>
<td>0.0031</td>
<td>4.39</td>
<td>0.82</td>
</tr>
</tbody>
</table>
Table 5.6: Accuracy of de-anonymization. % deanonymized refers to the the percent of nodes that received a mapping from the SDA and % correct refers to the percent of nodes mapped that were correctly mapped.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Method</th>
<th>% deanonymized</th>
<th>% correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fb107</td>
<td>Edge Swap</td>
<td>98%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>k-anonymity</td>
<td>98%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>NetGAN</td>
<td>6%</td>
<td>68%</td>
</tr>
<tr>
<td>Ca-GrQc</td>
<td>Edge Swap</td>
<td>52%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>k-anonymity</td>
<td>50%</td>
<td>99%</td>
</tr>
<tr>
<td></td>
<td>NetGAN</td>
<td>2%</td>
<td>67%</td>
</tr>
</tbody>
</table>

the best chance at revealing nodes. For k-anonymity, our $k$ threshold was set to 3 and for edge swapping, we selected 20% of existing edges to be swapped. The effectiveness of the SDA was measured by how many nodes were de-anonymized as well as what fraction of the nodes were correctly de-anonymized.

Table 5.6 shows the results of these experiments. It appears that GANs performs better in terms of creating graphs that resist de-anonymization attacks. However, this comes at a large cost to utility preservation. Figure 5.1 shows the resulting graphs from each method and what changes they invoke on the original graphs. From the visuals it shows that GANs disrupts the original architecture substantially and instead homogenizes the network, that is, brings the degree distribution closer to a normal distribution as it appears in Figure 5.2. As a result, nodes become more difficult to identify by their distribution, similar to how k-anonymity functions, however, the effects are more disruptive through NetGANs. This can be seen in Figure 5.2. The resulting graphs generated by NetGANs shows that in its current state it cannot anonymize graphs while still preserving utility.

5.4 Temporal Walks

Another method was tested for anonymizing graphs. Rather than training a model for each snapshot of a graph it would be more convenient to train a single model for the entire process. This model would need to learn the entire diffusion process and because the
Figure 5.1: Anonymized versions of the Fb107 graph next to the original.

Figure 5.2: Degree distributions of original and synthetic graphs. Both synthetic distributions are shifted to the right and flattened which leads to more a homogeneous degree distribution.

generator struggled to create attributed walks a restriction was imposed to keep the walk vectors 1-dimensional.

A possible solution to this task could be temporal walks. Given a diffusion process over a graph, such as behavior adoption, nodes can have an attribute with values 0 or 1. Nodes start with 0 and switch to 1 if influenced at some timestep $t$ by their neighbors. Only an initial starting node can have a spontaneous adoption, that is, it has no neighbors already exhibiting the behavior when choosing to adopt. Given this restriction and the fact that every node has a time-of-adoption (TOA) at $t$, we can embed the diffusion process into
temporal walks. A temporal walk that follows a diffusion process is a walk wherein adjacent nodes in the sequence have adopted behaviors at consecutive timesteps. If node $u$ is the first node in a walk and node $v$ is the second, then node $u$ had a $TOA = 0$ while node $v$ had a $TOA = 1$. The walks can be of uneven length as the walks will follow the paths a diffusion process creates which could have dead-ends. The discriminator was modified to parse variable length walks, however, the generator was restricted to still producing even length walks equal to the size of the longest walk used for training. Presently there’s no workaround to fixed length generation and this may contribute to inaccurate synthetic walks.

The intuition behind these temporal walks is that a diffusion process spreads by some set of rules over the nodes of the graph. Given enough walks that follow the diffusion process, a generator that learns to mimic these walks could create synthetic walks that recreate the diffusion process. The main challenges to this are (i) how to aggregate the walks to recreate an attributed dynamic graph, and (ii) how to evaluate these synthetic walks. Compiling the walks into a transition matrix as in the original work can provide the probabilities of a node infecting another node yet it cannot say at what timestep. In NetGAN, accuracy is measured by edge prediction, but this tests how well the generator learns a graph’s structure. We need instead a metric to measure the similarity between two sets of sequences.

A possible way to measure walk similarities is to embed the nodes within the walks using the Word2Vec technique [32], then performing principal component analysis (PCA) and viewing the resulting clusters. By embedding the walks with Word2Vec, new lower-dimensional vectors are created for each node in the graph based off the neighbors they appear next to in the walks. With PCA, the dimensionality reduction is taken a step further to allow for clustering of the node embeddings over their principal components. To start, we create three sets of walks: (i) the original temporal walks, (ii) a set of synthetic walks created by a trained generator and lastly, (iii) a set of baseline walks that do not obey time nor the path of diffusion. With these three sets of walks we use word2vec on each set separately to obtain an embedding of nodes from three different contexts. Finally we
perform PCA on these embeddings. Ideally, embeddings from synthetic and original walks should have high overlap with little overlap to the baseline. A hundred graphs were created using netDiffuseR with each have a set of sampled temporal walks. A model was trained on each set to create corresponding synthetic temporal walks.

Results indicate that the generator struggles to reliably create walks similar to the original. Every model experienced diverging loss between the competing networks as seen in Figure 5.4. Despite all having nearly the same loss, the resulting PCA’s varied substantially but most PCAs had little overlap between original and synthetic clusters. Example PCAs are shown in Figure 5.3. Across the plots, nodes could either be highly varied in their embeddings or highly clustered. In most of the plots the synthetic walks tended to always form a distinct cluster which also gravitated more to the baseline embeddings than the real. Only in a few cases, like in Figure 5.3c, could the generator come relatively close to the real embeddings which were a highly clustered instance. Unfortunately, this method of comparing similarity between sets of sequences is too indirect as it requires training a model to develop embeddings. Another method is needed, one that can operate directly on the sequences to output a metric such as cosine similarity but for sets. Given this shortcoming, the conclusion only suggests that as things stand, the generator struggles to mimic the temporal walks it was trained on.

After all the evaluations and attempted changes to NetGANs it seems this architecture may not be the best approach to anonymizing graphs, producing attributed graphs, or even real-world networks as its base architecture struggles to create quality graphs with significant variance between them. When attributed walks are used, the generator struggles to match the distribution of attributes by a large margin. Attempting a different approach altogether and using temporal walks to maintain 1-dimensional walks lead to poor results, as well. While the original paper showed promising results, extensive analysis shows that NetGANs may not be ready yet for adoption as a practical generative method and that deep learning generative models are still a developing technology.
Figure 5.3: Sample PCA analyses for the synthetic scale-free graphs. Ideally, both real and fake should cluster together closely while still being separate from random. On the contrary, results indicate no discernible pattern behind the different types of temporal and random walks.
Figure 5.4: A sample instance of temporal walk losses. Every model exhibited this loss pattern when trained on the scale-free graphs.
Chapter 6: Evaluation of GCN for Social Forecasting

In this chapter another uncommon task for graph learning is examined. This time the objective is to forecast user-activity on social media. The capability to forecast presents a new challenge by requiring an ability to predict and generate the same type of features or information used to make the prediction. Before this anonymization, we first study the capabilities of a forecasting model to see if it may function well over social media networks. The model we chose incorporates a GRU layer whose matrices have been replaced with GCN operations to leverage both temporal and network structure properties to forecast an arbitrary length of steps. We perform experiments on two real-world datasets and compare the model to some classical regression models.

6.1 Problem Definition

We aim to forecast user activities on two social media platforms. Specifically, we aim to be able to forecast the Push activities on the repositories in our GitHub dataset and the retweets as would happen in the White Helmet conversations on Twitter. The machine learning task is thus the following: given the history of user activities on a platform, predict a sequence of future activities, their authors and timing, for a given period of time.

This problem is motivated by the significant social challenges that surfaced due to the heavy, unregulated, and often unethical use of social media platforms. For example, the fear of disinformation spreading in the aftermaths of the Sri Lanka Easter 2019 bombing that killed hundreds of people prompted the Sri Lankan government to disable social media platforms for weeks. Predicting how information spreads in such a situation may enable authorities to control such risks with more targeted intervention campaigns, such as blocking
key accounts responsible for the wide spreading of information (not necessarily the content producers), or inoculating the population with verifiable, accurate information before they are flooded with false information.

In order to address this problem, we model our data as dynamic graphs with static nodes and edges and temporal node attributes. We chose a static graph structure in order to conform with the current implementation of deep learning frameworks that include graph convolutions which are briefly described in Chapter 3. Realistically, a social network has new users joining and new edges forming continuously, but because of restrictions imposed by GCNs, this aspect needed to be ignored.

Let a graph be represented as $G = \{V, E\}$ where $|V| = N$ and $E$ is a set of edges. Every node $v \in V$ is either a software repository (for GitHub) or a user (in Twitter). The temporal attribute for each node is their activity or signal over time. Let $X^t \in \mathbb{Z}^N$ be the graph signal for $G$ observed at time step $t$. Specifically, $X^t(i)$ represents the number of activities of user $i$ during time interval $t$. Forecasting user actions in a social network can then be represented by learning a function $h$ that maps $\delta$ past signals to $\delta$ future signals, given $G$ and $X$:

$$[X^{(t-\delta)}, ..., X^t; G] \xrightarrow{h()} [X^{(t+1)}, ..., X^{(t+\delta)}]$$

Our objective is to train a deep neural network that learns this mapping function. A model that is capable of forecasting activities can effectively simulate the diffusion processes in the underlying network.

### 6.2 Challenges

The challenges discussed in detail in this section can be summarized as follows: while we have big data (in number of events), it appears sparse when modeled over time at practically useful time granularity. Our datasets are from different social media platforms: in Twitter reactions to a post fire up rapidly, due to user engagement and ease of responding.
In GitHub, user contributions require more effort and time, and thus the activity per user may be less frequent. Yet despite the distinct contexts of our datasets, the time series for interarrival times and activity rates are highly similar and thus pose the same challenges to learning algorithms.

We conducted experiments on two graphs modeled from real-world social interactions on Twitter and GitHub, as presented above. In our problem definition, the task is regression of user activity. For each user-signal, we aim to predict a vector of the next $T$ time steps given a vector of $T'$ historical time steps from the signal. User signals were discretized into daily counts and then grouped into weekly series to predict weekly patterns of users. Daily granularity was chosen as it provides the best balance between having enough training records while still containing strong enough signals that can be learned by a model. We also train models on different aggregated instances of the graph to explore potential solutions to the problems inherent in dynamic user activity in social media. Due to scalability, a known issue regarding GCNs [11], no DCRNN model could be trained over the first graph we constructed for the GitHub records.

For all DCRNN models, we use the same hyper-parameters. We trained for 500 epochs with an early stopping of 50. The model consisted of one DCRNN layer with 32 hidden units in both the encoder and decoder. We chose Adam [22], a popular adaptive learning rate optimizer, and a starting learning rate of $1e^{-3}$. For both datasets, we split the time series samples temporally into 70% training, 10% validation, and 20% testing. In the White Helmets dataset, the train and validation period comprised data from 04/01/2018 to 02/10/2019, and the test period ranged from 02/11/2019 to 04/30/2019. For the CVE dataset, the train and validation sets consisted of data from 01/01/2015 to 08/06/2017, and the testing was from 08/07/2017 to 03/31/2018.

Finally to evaluate DCRNN performance, we compare it against the following time series regression baselines: Historical Average (HA) models the time series as a seasonal pattern, and uses past seasons weighted average to inference next time step predictions.
Vector Auto-regressive model (VAR) is a widely used forecasting method that attempts to fit a linear function over past values of a variable to predict its future time steps. The static method is a naive time series shifting approach which assumes that activity at some time $t+1$ will remain the same as activity at time $t$. Root Mean Squared Error (RMSE) was chosen for evaluating the accuracy of every model and graph instance. In our attempts to solve the task of forecasting user-activity, several challenges arose which are discussed in detail below. Despite these challenges DCRNN still outperformed the baselines across most instances of our datasets and the accuracy of every model predicting out to the 7th day is shown in Table 6.2.

### 6.3 Sparsity

The daily counts for users are almost all zero entries or no-activity. For Twitter we found that 95% of records were zero and 99% for GitHub. To illustrate, Figure 6.1a shows the weekly activity rates for users across both platforms where we see that more than of 70% users have a weekly activity rate of 0. This zero-inflation can be seen in other works of deep learning on social networks [36, 29] denoting that this zero-inflation characteristic is not unique to our datasets. However, it has also been shown that users on social platforms operate in a heterogeneous and bursty fashion, which produces spikes within relatively flat activity levels [4, 18]. We confirm this occurs in our dataset by observing the distribution of inter-arrival times of retweets and push events as is shown in Figure 6.1b as well as the weekly activity rates of users across datasets. More than 70% of users have a weekly activity rate 0 but with most inter-arrival times being so short we can infer that users typically bunch their activities together. The major sparsity of data with occasional spikes poses a major obstacle for forecasting as it encourages models to severely under predict values.

To demonstrate, models were built for both White Helmets and CVE datasets to predict the weekly pattern of every user in the networks. As previously stated, the DCRNN model outperforms most model configurations we explored. However, such high performance
Figure 6.1: Weekly activity rates and distribution of inter-arrival times. $P(t)$ denotes the density of time $t$. To create $P(t)$, both datasets were aggregated into bins which spanned a logarithmic scale to account for the rapid decay in frequency of growing inter-arrival times.

is assumed to be from the abundant zero records. Thus, the models tend to predict lower levels of activity and fail to capture spikes, as shown in Figures Figure 6.2a and Figure 6.2b.

These behaviors, however, are not the goal of forecasting user activities. Ideally, we want to simulate the spikes present in user signals accurately. One can choose to filter out and focus on "active" users from their dataset, users whose activity patterns can be learned, but care must be taken defining such active users. For example, there can be the trivial case where some users make a single retweet who can be considered "non-active", but there is the more likely case of a user who cyclically retweets several times in a day, stays relatively silent for a few days or weeks, and then produces another set of retweets. Such is the case for Figure 6.2a where 2 weeks separate the days when a user retweets the most.

This bursty nature means filtering by activity rate is unreliable due to how rapidly rates can change. Although a user with a high constant rate of activity may seem suitable for a deep neural network, they are not representative of the actual user base of social networks. To observe the effect of filtering users, we first define non-active users as any user or repository whose average daily count was less than the global average and train a model on the active users’ time series. The corresponding graphs are also updated to remove non-active nodes (users). Although this model only takes into account the active users, the
resulting time series were again relatively sparse. As one might expect, the accuracy worsens when filtering by active users who produce slightly more complex signals than inactive.

6.4 Granularity

Another approach to remedy the large data imbalance in user activities is to change the time granularity of the forecast instead of filtering users based on activity rates. The intuition is that at higher time granularities, aggregated user activity produces more stable or noticeable signals. However, this approach comes at the cost of training data. Deep Learning models work best with a large amount of data to train and so increasing the time granularity will still lead to a model with poorer performance due to the observation period not being sufficiently long. This was the case when we built models on a monthly granularity, where results were too poor to include.

Alternatively, one may cluster users into communities and aggregate activities by community. Though we can no longer forecast activities at the user-level, we may have a meta-signal whose patterns are easier to learn. To test this idea, we clustered users in the White Helmets datasets using the Louvain clustering method [8]. For clustering in GitHub recall that every repository has an associated CVE(s) and subsequent corresponding product(s) and vendors(s). We cluster GitHub repositories by products or vendors resulting in two new graphs where nodes are now clusters of repositories tied by a product or vendor, and edges form between clusters if a CVE is shared between them. We aggregated the user-signals contained in each cluster into a single larger signal through summing record counts from the same day and we trained models to predict these meta-signals. The new graphs are constructed by treating clusters as nodes and an edge exists between clusters if the nodes between clusters shared an edge in the original graph. We also experimented with combining aggregation with filtering on both networks but these too were met with poor performance. Table 6.1 shows the basic network characteristics for every approach we experimented with.
Table 6.1: Graph characteristics for each instance of the two networks. The average user activities are presented at the daily granularity.

<table>
<thead>
<tr>
<th>Model</th>
<th>Nodes</th>
<th>Edges</th>
<th>User Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>WH</td>
<td>6,376</td>
<td>96,316</td>
<td>0.16</td>
</tr>
<tr>
<td>WH-Active</td>
<td>1,247</td>
<td>33,655</td>
<td>0.42</td>
</tr>
<tr>
<td>WH-Cluster</td>
<td>15</td>
<td>103</td>
<td>46.79</td>
</tr>
<tr>
<td>Github</td>
<td>22,052</td>
<td>8,438,619</td>
<td>0.007</td>
</tr>
<tr>
<td>GitHub-Product</td>
<td>1,391</td>
<td>14,566</td>
<td>0.31</td>
</tr>
<tr>
<td>GitHub-Product-Active</td>
<td>199</td>
<td>5,343</td>
<td>2.25</td>
</tr>
<tr>
<td>GitHub-Vendor</td>
<td>498</td>
<td>2,047</td>
<td>0.62</td>
</tr>
<tr>
<td>GitHub-Vendor-Active</td>
<td>66</td>
<td>1,032</td>
<td>4.84</td>
</tr>
</tbody>
</table>

We found that predicting over meta-signals degrades performance across every approach even further as seen in Table 6.2. The White Helmets Daily and active user models both outperform the clustered version. However, despite degraded performance, we see the model makes better attempts at matching meta-signals as seen in Figure 6.2d. At first glance, it may appear the model attempted to merely shift its input sequences by a day to create the output, but minor deviations from ground truth suggest that the model tried to learn some pattern to the series. Similar results can be seen in Figure 6.3d where the model could capture the relative period of dormancy and subsequent rapid bursts but overall struggled to match the magnitude of such spikes opting instead for smoothing the jagged signal.
Figure 6.2: Sample forecasting plots for models trained on White-Helmets.

Table 6.2: Weekly activity forecasting performance for multiple models. In order the models are: Historical Average (HA), Vector Auto-Regressive (VAR), static shifting (STATIC), and Diffusion Convolutional Recurrent Neural Networks (DCRNN).

<table>
<thead>
<tr>
<th>Model</th>
<th>Period</th>
<th>Metric</th>
<th>HA</th>
<th>VAR</th>
<th>STATIC</th>
<th>DCRNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>WH-Daily</td>
<td>7 Days</td>
<td>MAE</td>
<td>1.20</td>
<td>1.71</td>
<td>1.68</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RMSE</td>
<td>2.21</td>
<td>2.67</td>
<td>2.70</td>
<td>2.07</td>
</tr>
<tr>
<td>WH-Daily-Active-Users</td>
<td>7 Days</td>
<td>MAE</td>
<td>1.44</td>
<td>1.94</td>
<td>1.81</td>
<td>1.94</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RMSE</td>
<td>2.46</td>
<td>3.06</td>
<td>2.96</td>
<td>2.93</td>
</tr>
<tr>
<td>WH-Daily-Cluster</td>
<td>7 Days</td>
<td>MAE</td>
<td>73.86</td>
<td>62.88</td>
<td>42.80</td>
<td>15.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RMSE</td>
<td>127.29</td>
<td>111.97</td>
<td>98.41</td>
<td>56.71</td>
</tr>
<tr>
<td>GitHub-Product</td>
<td>7 Days</td>
<td>MAE</td>
<td>7.20</td>
<td>22.93</td>
<td>9.49</td>
<td>1.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RMSE</td>
<td>28.83</td>
<td>73.27</td>
<td>38.24</td>
<td>7.07</td>
</tr>
<tr>
<td>GitHub-Product-Active</td>
<td>7 Days</td>
<td>MAE</td>
<td>10.60</td>
<td>N/A</td>
<td>13.48</td>
<td>9.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RMSE</td>
<td>36.18</td>
<td>N/A</td>
<td>47.97</td>
<td>36.87</td>
</tr>
<tr>
<td>GitHub-Vendor</td>
<td>7 Days</td>
<td>MAE</td>
<td>10.71</td>
<td>91.09</td>
<td>12.81</td>
<td>10.27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RMSE</td>
<td>35.94</td>
<td>220.46</td>
<td>45.47</td>
<td>36.34</td>
</tr>
<tr>
<td>GitHub-Vendor-Active</td>
<td>7 Days</td>
<td>MAE</td>
<td>15.03</td>
<td>19.84</td>
<td>17.68</td>
<td>6.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RMSE</td>
<td>44.22</td>
<td>53.93</td>
<td>55.90</td>
<td>29.92</td>
</tr>
</tbody>
</table>
Figure 6.3: Sample forecasting plots for models trained on GitHub.
Chapter 7: Conclusion

The studies presented in this paper were motivated by the recent successes in deep learning across a multitude of fields. Seeing the growth of graph learning as one such field, we attempted to expand upon the list of tasks models are capable of performing regarding graphs. In the first study, we took an existing model for generating graphs, one which utilizes random walks and GANs, and evaluated its ability to anonymize graphs and produce attributed ones. In the second study, we take an existing model for traffic forecasting, one which utilizes GRU and GCNs, and experiment with whether it can predict user activities over two real-world datasets taken from Twitter and GitHub. Across the two studies we make several contributions in the space of graph learning:

- This thesis is the first attempt to anonymize graphs using deep neural networks and contains comparisons to traditional anonymization methods.

- We perform an extensive analysis on the ability for NetGAN to generate quality synthetic examples.

- We identify potential issues behind random-walk based approaches to learning topologies of graphs.

- Demonstrate issues in attempting to learn temporal walks as well as issues in embedding a diffusion process into walks

- We identify data characteristics which make DNNs unsuitable for time series forecasting.
7.1 Summary on Graph Anonymization

Graph anonymization has been shown to be a difficult problem as a delicate balance between utility preservation and vulnerability must be struck. Likewise, DNN based graph generative methods are still a developing technology and may be unsuited for a difficult task like anonymization. In this work we experimented with a DNN based graph generator to observe its potential to create anonymized attributed graphs.

In attempting to produce attributed graphs, a discriminator could successfully identify whether walks were real or fake across different steps of a diffusion. When generating attributed walks, however, results show poor performance as the generator cannot correctly label nodes nor match the distribution of adopters.

After evaluating attributed walk generation, the original architecture was re-examined. We performed experiments on the model’s ability to create graphs which preserve utility and create varying synthetic examples. Our results in these experiments show that the model does not actually match metrics well either when compared to more classical generative methods such as DK-2 generation. In particular, producing nodes of degree 1 is a weak point of NetGAN, but this may be a consequence of the assembly procedure utilized. When inspected visually, synthetic graphs of real-world networks appeared to have been homogenized, wherein community clusters were dissolved. In addition, degree distributions appeared to shift slightly to follow a normal distribution rather than typical heavy tailed distributions. The synthetic graphs also lack variance as shown through the D-measure, a metric used to calculate similarity between graphs. Synthetic examples were shown to have distances close to 0 and nearly equidistant from the original. The issue with variance may be tied to the graph assembly portion of the architecture as edges are created by sampling from a transition matrix. By both crafting the transition matrix from a large set of synthetic walks as well as sampling a large set of edges from the matrix, synthetic graphs typically exhibit the same topology amongst each other. When measuring vulnerability, the network demonstrates a high resistance to an SDA compared to more classical techniques as natural consequence
of an inability for utility preservation. Trying a different approach where attributes were embedded into temporal walks that follow the boundaries of a diffusion process, we measure the similarity of generated walks to real ones and compared the distance to random walks. Results again show that the model cannot produce realistic temporal walks.

7.2 Summary on Social Forecasting

In recent years, CNNs have demonstrated excellent performance in a variety of tasks including image classification, action recognition, and natural language processing. However, these architectures have been shown to be exclusively suitable for tasks involving Euclidean spaces. Such limitation inspired the implementation of GCNs which also aim to leverage the idea of convolutions for neighborhood feature aggregation to capture spatial dependencies for non-Euclidean domains, often characterized as complex graphs. GCNs, when used in recurrent architectures, have shown state-of-the-art performance in traffic forecasting tasks [26, 50] and were mathematically proven to be generalizable.

This success and generalizability motivated our study to apply spatio-temporal GCN models in online social networks, and investigate whether the same level of success for forecasting user behavior over time could be attained. Our experiments show that predicting future user activity levels on social media platforms is quite challenging, and the solutions employed in apparently similar problems, i.e., traffic forecasting, are not applicable to these new platforms. This may be due directly to the heterogeneity of user activity in time and space. Specifically, most users tend to be inactive over long intervals, or perform relatively few activities overall. Interestingly, despite the differences in the social media platforms and the contexts (discussions about the Syrian Civilian Search and Rescue Group vs. announcements of Software Vulnerabilities), similar user activity patterns are observed as been shown in past studies of social networks. The historical time series for each individual user is too sparse, which essentially affects the model’s learning process. A model trained on these sparse signals will likely prefer to predict 0, as this is the most common volume of
activities for a day which leads to very high accuracy but poor performance. As an attempt to alleviate this issue, we experimented with different time granularity aggregations, including daily, weekly, and monthly levels. We discovered that aggregating data beyond daily granularity significantly reduced the total number of samples and the resulting data was not enough to fit a time-series model. Despite worsening accuracy, the models in which aggregation was conducted still yielded forecasts whose signals better match ground truth values. Furthermore, despite poor performance, the DNN still had lower error values than traditional models.

7.3 Future Work

While NetGAN may have proven unsuitable, we believe it still has potential to be a powerful graph generative method. The point of failure seemed to be in the graph assembly which in the original paper was a sampling over a transition matrix. Future work could include updating the graph assembly algorithm to yield graphs with more variance in their metrics. Alternatively, exploring other generative methods may yield better results for graph generation such as in [49] which utilizes RNN layers without a GAN architecture. In regards to forecasting, we believe a point of future work could be in modeling the network from which the time series were collected. Having an incorrect graph representation when forecasting can hurt performance as the feature aggregation would result in feature maps that do not provide the model with any spatial dependency information.
References


