Robustness in Link Prediction

William Howard-Snyder
Department of Computer Science
University of Washington
Seattle, WA 98103
howarwil@cs.washington.edu

Pallavi Banerjee
Information School
University of Washington
Seattle, WA 98103
pallavib@uw.edu

Therese Pena Pacio
Department of Computer Science
University of Washington
Seattle, WA 98103
tpacio@cs.washington.edu

Hans Easton
Department of Computer Science
University of Washington
Seattle, WA 98103
hans00@cs.washington.edu

Abstract

In this project, we aim to explore a variety of methods for link prediction on large, noisy, and incomplete graphs.

1 Introduction

Graphs are used to model a wide variety of complex systems ranging from social networks with billions of users, to protein-drug interactions that are vital for understanding the effects of a treatment. Often, our knowledge about how the components of these systems are related is incomplete. For example, a user's negligence of their social media might lead to outdated or accidental friendships; the cost of wet lab experiments prevent biologists from exploring all combinations of protein-drug interactions.

Given the importance of graphs in the big data era, we want to explore methods for link prediction and how these methods are affected by varying degrees of noise. We are particularly interested in Graph Neural Networks (GNN), but also seek to understand non-deep approaches such as matrix completion and simple node proximity measures. In this project proposal, we summarize and critique current approaches for link prediction, and outline our experimental procedure for how we intend to benchmark our selected methods.

2 Reaction

In this section, we summarize and critique several papers related to link prediction. The first is [3], which provides a survey of traditional methods for link prediction. The second paper [4] presents a survey of Graph Neural Networks (GNN), which have recently emerged as the method of choice for modeling graph-structured data. The last paper [1], introduces GraphSAGE, which is a particular GNN that makes architectural and training choices that allow it to scale better to large graphs than previous methods.

2.1 Summary

In [3], Liben-Nowell and Kleinberg investigate whether a graph's evolution can be modeled with just the features intrinsic to the graph itself (i.e., topology of the graph). In doing so, they explore several measures for node proximity and evaluate their efficacy at predicting future node-node interactions.

Preprint. Under review.

Specifically, at training time, they are given a graph G and learn a function score(x, y), that measures "how close x and y are in G". At inference time, they rank the pairs based on score, and output the most similar nodes that are not already in the graph. They call this problem *link-prediction*.

In this work, Liben-Nowell and Keinberg perform rigorous experiments on citation networks to determine which methods yield the best performance. They find that neighborhood-based approaches, work surprisingly well. These approaches operate under the assumption that vertices will be more likely to have a link between them if there is large overlap in their sets of neighbors.

Among these best performing methods were Adamic-Adar Distance and Common Neighbors. Let $\mathcal{N}(x)$ be the neighbors of x, and $\texttt{score}_{CN}(x, y)$ denote common neighbor score, and $\texttt{score}_{AA}(x, y)$ denote Adamic-Adar similarity. Then, they define

$$\texttt{score}_{CN}(x, y) := |\mathcal{N}(x) \cap \mathcal{N}(y)| \qquad \texttt{score}_{AA}(x, y) := \sum_{z \in \mathcal{N}(x) \cap \mathcal{N}(y)} \frac{1}{\log |\mathcal{N}(z)|}$$

Common neighbors is quite intuitve as it is just the number of neighbors shared by x and y. Adamic-Adar Similarity is slightly more complex, but it is very similar. Instead of taking the raw count of shared neighbors, it weights the neighbors by their rarity (i.e., two nodes sharing many neighbors of degree 2, is more meaningful than sharing neighbors that have many neighbors).

More recently, the research community has shifted to *learning* node similarity functions with GNNs directly from the data, rather than using hand-crafted heuristics. GNNs try to learn representations of nodes that encode information about the structure of the graph. The intuition is that by leveraging the topological relationships between nodes we can get higher-quality representations that yield better performance on downstream tasks (e.g., node classification, recommendation, etc.). To compute similarity between nodes (i.e., whether an edge should be present), it is common to measure the similarity between the *hidden representation* of the two nodes.

In [4], Wu et. al discuss a wide range of Graph Neural Networks for learning node/link/graph level representations and categorize them into four overarching methods: Recurrent GNNs, Convolutional GNNs, Graph Autoencoders, Spatio-Temporal GNNs. For this project, we are primarily interested in Convolutional GNNs (ConvGNNs) as they are flexible, efficient, and naturally extend to the task of link prediction.

ConvGNNs are neural networks that extract node representations by aggregating the representations of neighbors via message-passing. This is typically done by stacking a fixed number of *convolutional layers*. Wu et. al identify two categories of ConvGNNs:

Spectral ConvGNNs define convolutions in terms of graph signal processing. These methods compute the eigenvectors of the normalized Laplacian defined as $L = I - D^{-1/2}AD^{-1/2}$, where D is a diagonal matrix of node degrees. Since L is symmetric, we can factor it as $L = U\Sigma U^\top$, where the columns of U are the eigenvectors of L and Σ is a diagonal matrix of corresponding eignevalues. The kth layer of a Spectral ConvGNN is

$$h_v^{(k)} = U \left(U^\top h_v^{(k-1)} \odot U^\top W^{(k)} \right)$$

where $W^{(k)}$ is a matrix of learnable parameters called the filter. This operation can intuitively be understood as mapping the previous layer's representation, $h_v^{(k)}$, and the filter, $W^{(k)}$, into the graph's "natural basis", multiplying the result together, and then mapping it back into the original basis. The parameters for each filter are learned so as to extract certain aspects of the graph much like how convolutional filters in CNNs emphasize certain aspects of an image.

Spatial ConvGNNs Generalize the convolutional architecture in computer vision, which operates on grid graphs, to graphs of arbitrary structure. These networks are composed of convolutional layers that iteratively compute a nodes "representation" based on that node's features and those of its neighbors. In [5], Xu et. al formally define a framework for the kth layer

$$a_v^{(k)} = \texttt{AGG}^{(k)} \left(\left\{ h_u^{(k-1)} \ : \ u \in \mathcal{N}(v) \right\} \right), \quad h_v^{(k)} = \texttt{COMBINE}^{(k)} \left(h_v^{(k-1)}, a_v^{(k)} \right)$$

where $a_v^{(k)}$ is an aggregation (e.g., sum, concatenation, mean) of node representations and $h_u^{(k-1)}$ is the representation of node u for the kth layer, which is a function (e.g., MLP) of the aggregation

step. Typically, $h^{(0)}$ is one-hot encoded, or node features. Broadly, Spatial ConvGNNs all fit into this form.

One particular GNN method that has recently seen a lot of success in link prediction is [1]. This is a Spatial ConvGNN, so it fits the `AGG` and `COMBINE` scheme described above. In this work, Hamilton et. al propose sampling from a nodes neighborhood in the aggregation function, rather than aggregating across all neighbors of a node. This allows for fixed batch sizes, and make training significantly more efficient.

2.2 Critique

Graph neural networks are computationally costly. Training requires saving the entire graph data and intermediate states of all nodes into memory [4]. Various implementations of GNNs propose different ways of optimizing the training process. GraphSage, a particular model we are interested in, uses a recursive sampling method to sample trees within K steps rooted at a node n [1]. Another method, Fast Learning with Graph Convolutional Networks, improves on this method by sampling a fixed number of nodes for each convolutional layer rather than a fixed number of neighbors per node. However, this method risks sparse connections between layers as nodes are sampled independently between layers.

GraphSAGE implements a spatial-based Convolutional Graph Neural Network as it leverages node features to learn embedding functions rather than matrix factorization approaches used with spectral-based Convolutional Netoworks. This method allows the model to learn topological structure of each node's neighborhood as well the distribution of node features in the neighborhood. This is key for GraphSAGE's demonstrated high performance on predicting links for unseen nodes since factorization techniques constrain the embedding of new nodes to the function optimized during training. By using features of the neighborhood, GraphSAGE learns features that meaningful represent the graph's topological structure. However, as [4] points out, the mathematical foundations of spatial convolutional GNNs are more dubious than those of spectral methods. Spectral methods assume a graph to be undirected graph and can be represented with a Laplacian matrix, which serves as the basis for rigorous proofs on the embedding functions.

564Further, GraphSAGE theoretical proofs makes an assumption that each node has a unique feature representation, therefore each node can be mapped to an indicator vector and identify node neighborhoods. This assumption may not hold as we work with high-dimensional datasets that we may hope to optimize by excluding features. As we exclude features, we may see that nodes may not be unique in their feature representation and this assumption would no longer hold.

Finally, the use of a deep learning model like a Graph Convolutional Network raises concerns on the interpretability of the link predictions. Understanding how the model generated its link prediction given graph data may be difficult due to the high complexity of these black box models. This may become a concern as we apply these models to other domains such as drug discovery, which may require an in-depth interpretation of the model's learned parameters to ensure the efficacy of these drug linkages. Non deep learning methods of characterizing node similarity proposed by [3] may be more appropriate for problems in these domains. These statisitcal metrics provide a more intuitive understanding of node similarity using information such as Jaccard Similarity and distance despite lower performance.

2.3 Brainstorming

Paper [3] points out few interesting methods to improve upon the graph neural network performance by improving the training data quality. For example, If the graph used in the training phase is sparse and not bipartite, then we may expect improvement in performance metrics if the input graph is converted to a bipartite graph. Another way to improve upon baseline graph neural network in link prediction use-cases is to assign more weight to latest/recent interactions. By doing this we may be able to make the predictions more tailored to the dynamic trends in either user preferences or market interactions (depending on the use case). However, due to the limited time we choose to not explore the area of improving graph input quality.

In paper [4] Wu et. al mention the scalability and model depth drawbacks of using a graph neural network. In order to achieve scalability, we ideally sample/cluster the graph which may lead to

removal of important information. Understanding the trade off between scalability and graph integrity has been identified as one of the key directions forward. In order to understand the trade off, we may look at the performance of the graph neural network on different subsections of the data. This will help inform us about the trade off between scalability and information loss.

Further, [4] mention how a ConvGNN performance drops dramatically with an increase in the number of graph convolutional layers. This is because the convolutions push embedding of adjacent nodes closer to each other, resulting in all nodes' representations to converge to a single point for very deep ConvGNN. As raises the question of whether going deep is still a good strategy for learning graph data, we propose to look at non- deep learning algorithms and benchmark their performance againgst a ConvGNN performance to understand if there is a significant loss when using ConvGNN or if non-deep learning algorithms perform as well.

One element that is consistently missing from the literature we've reviewed is explicitly considering the noise present in these graphs. Real graph data is large, incomplete, and has potentially many false associations, especially in the biomedical setting. We would like to better understand how these methods perform under varying degrees of noise.

3 Proposal

In our project we will investigate various methods for link prediction in large, noisy, biomedical networks. Specifically, we want to identify missing links within a network, using observable data. Note that we are not attempting to model the temporal component as Liben-Nowell and Kleinberg do in [3].

3.1 Description

Our project will compare and contrast different methods for solving the missing link problem. Specifically, we will implement and benchmark established algorithms discussed in the Reaction section. We will experiment with both non-deep-learning-related methods and GNNs. In addition to benchmarking on the original dataset, we will also empirically verify the robustness of these methods by varying the degree of missing links in our dataset. For example, we will investigate how prediction accuracy changes for each method when fake edges are introduced into the data or when only a subset of existing edges are chosen. Through these experiments, we hope to gain insight into the benefits and drawbacks of each method on the dataset.

3.2 Project Outline

Here is a rough outline of the work we will do for our project in chronological order.

Data Wrangling We will load and prepare the OGB data into our local development environment. We may be able to use the PyTorch Geometric and DGL data loaders that handle dataset downloading and standardized dataset splits provided by the OGB package.

Baseline Implementation We will study the non-deep-learning methods for solving the missing link problem described in [3]. We will implement the highest performing methods listed in the paper and run a set of benchmarking processes on the drug-drug interaction dataset described below.

GNN Implementation Once we've established working baselines, we will focus on implementing the graph neural networks, and preparing our benchmarking pipeline. We will study and implement relevant learning algorithms discussed in the other two reaction papers on graph neural networks.

Robustness Analysis After implementation and benchmarking, we will do a comprehensive analysis of how the algorithms perform on the dataset and discuss the tradeoffs for each algorithm. We will also explore how the accuracy of the algorithm prediction changes on variations of the dataset. For example, if we gradually remove a subset of nodes or edges in the original dataset, will the accuracy go down linearly for each algorithm as the nodes decrease linearly? If not, what is the correlation between the two variables? We can then plot graphs and draw conclusions based on these

experiments, and our hope is that there are additional insights we can derive for each algorithm with the variations of the dataset.

Improvements Finally, with the insight we gain in the previous step, we hope to make minor improvements to promising implementations for the missing link problem on the drug-drug interaction dataset by incorporating prior knowledge of the dataset. Since each drug node in the drug-drug interaction network is mapped to a unique drug ID in DrugBank, we might be able to draw scientific insight from the model's predictions and potentially augment the given graphs with richer information. However, this might not be feasible and is constrained by the amount of additional work and computing resources we are given.

3.3 Data

For our project, we will be utilizing an OGB dataset that was compiled by researchers at Stanford University [2]. OGB is a collection of benchmark datasets that are diverse, challenging and realistic, with the goal of promoting scalable, robust and reproducible research in graph machine learning. Specifically, we will use the `ogbl-ddi` dataset which is a homogeneous, unweighted, and undirected graph that represents the interactions between drugs. Each node in the graph represents a drug that is either FDA-approved or an experimental drug. The edges in the graph depict the interactions between drugs and can be interpreted as a situation where the combined effect of taking two drugs together is significantly different from the expected effect when the drugs are taken individually. The graph dataset comprises of approximately 4.5×10^3 nodes and 1.3×10^6 edges, making it a suitable dataset for running various missing link prediction algorithms.

Time permitting, we would also like to investigate performance on other biological/medical datasets from OGB such as `ogbl-ppa`. This dataset consists of a graph where the nodes are proteins from 58 different species, and edges represent a biologically meaningful associations such as physical interactions, co-expression, homology or genomic neighborhood [2]. This dataset has 5.8×10^5 nodes and 3.03×10^7 edges making it approximately two orders of magnitude larger than the drug-drug interaction one.

3.4 Methods

In the Reaction section we described several methods for link prediction. We will use these methods, with a couple of additions as summarized in Table 1.

Table 1: Methods

Type	Name	Description
Deep Learning	GraphSAGE [1]	Learn node representations via 2-hop subsampled graph neighborhood.
	SEAL [6]	Learn general graph structure features from local enclosing subgraphs.
Neighborhood	Common Neighbors [3]	Neighbors are similar to the extent that they share neighbors.
	Adamic-Adar [3]	Common neighbors weighted by inverse neighbor degree.
Matrix Factorization	SVD	Treat the eignevectors as node features and compare with cosine similarity.

3.5 Evaluation

We will evaluate how will each method predicts new association edges given the training edges. Following in the footsteps of [2], we will use the Hits@K metric to measure prediction performance. Specifically, for each edge in the test set, we will sample N negative edges (i.e., edges that are not in the graph). Then we use each method to rank all $N + 1$ edges by assigning a score indicating the confidence in an association between the two ends of the edges. Hits@K is the proportion of

test edges that are ranked as one of the top K edges. In [2], Hu et. al found that $N = 100,000$ and $K = 20$ worked well for this dataset.

Since we are particularly interested in the robustness of these methods, we will also randomly remove varying percentage of the nodes in the training portion of the graph (0%, 5%, 10%, 20%, 40%), train on the resulting edges, and measure performance (accuracy, hits@20, precision-recall, auroc, etc.) on test set. We will create plots with percent edges removed on the x-axis and performance on the y-axis and qualitatively evaluate how each method performs under varying degrees of noise.

Additionally, if some methods are much more sensitive to noise than others, we will investigate why by looking at which edges are misclassified to see if there are any patterns. This might involve doing a literature review on relevant biological entities (e.g., drugs/proteins).

3.6 Project Goals

Our primary goal in this project is to gain a deeper understanding of the problem of link prediction, and why certain models are more robust to noise. To this end, we will rigorously experiment with five prominent methods for link prediction, observing how prediction performance varies with perturbations of the dataset. We will also write a formal report summarizing our findings succinctly with clear and objective analyses. Finally, we will write significant code for benchmarking these methods that we intend to make public and highly readable (perhaps through the use of an interactive Jupyter Notebook). This facilitates replication of our methods, and contributes to the transparency of research in the graph modeling community.

References

[1] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017. URL `http://arxiv.org/abs/1706.02216`.

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

[3] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, page 556–559, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137230. doi: 10.1145/956863.956972. URL `https://doi.org/10.1145/956863.956972`.

[4] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019. URL `http://arxiv.org/abs/1901.00596`.

[5] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018. URL `http://arxiv.org/abs/1810.00826`.

[6] M. Zhang and Y. Chen. Link prediction based on graph neural networks. *CoRR*, abs/1802.09691, 2018. URL `http://arxiv.org/abs/1802.09691`.