論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Enhancing Aggregation Scheme in Graph Neural Networks to Improve Learning
著者(和文)	MAURYASunil Kumar
Author(English)	Sunil Kumar Maurya
出典(和文)	学位:博士(学術), 学位授与機関:東京工業大学, 報告番号:甲第12482号, 授与年月日:2023年3月26日, 学位の種別:課程博士, 審査員:村田 剛志,秋山 泰,岡﨑 直観,DEFAGO XAVIER,石田 貴士
Citation(English)	Degree:Doctor (Academic), Conferring organization: Tokyo Institute of Technology, Report number:甲第12482号, Conferred date:2023/3/26, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

TOKYO INSTITUTE OF TECHNOLOGY

DOCTORAL THESIS

Enhancing Aggregation Scheme in Graph Neural Networks to Improve Learning

---

Author: Sunil Kumar Maurya

Supervisor: Tsuyoshi Murata

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Department of Computer Science

School of Computing

TOKYO INSTITUTE OF TECHNOLOGY

Abstract

Department of Computer Science
School of Computing

Doctor of Philosophy

Enhancing Aggregation Scheme in Graph Neural Networks to Improve Learning

by Sunil Kumar Maurya

Graph neural networks (GNNs) are indispensable tools for learning on graph-structured data. One of the most important steps of GNN training is the feature aggregation step, where a node aggregate features from its neighboring nodes in the graph to capture the neighborhood structural information. The graph-structured data can come from a wide variety of fields, and many different types of learning tasks, like node classification, node ranking, link prediction, etc., can be performed on these datasets. However, often GNNs with the default aggregation scheme are used to learn on data, and domain-specific knowledge is not leveraged to improve the learning capabilities of the model. Hence, the use of the default aggregation scheme in GNNs may not always be optimal. In this thesis, the author proposes improved aggregation schemes relevant to two different graph learning tasks. The first task is node ranking, where the author proposes a shortest-path based aggregation scheme and introduces two GNN models, GNN-Bet and GNN-Close, to approximate Betweenness and Closeness centrality of the nodes, respectively. The second task is node classification, where first, the author empirically demonstrates the effectiveness of selective feature aggregation over default feature aggregation. Then based on the observations, the author proposes a GNN model FSGNN, which can automatically learn to select hop features enabling high classification accuracy on homophily and heterophily datasets. With extensive experimental results on both tasks, the author shows that the proposed GNN models with improved aggregation schemes outperform other state-of-the-art methods and GNN models.

Acknowledgements

Firstly, I would like to thank my thesis advisor, Professor Tsuyoshi Murata, for his kind support and opportunities throughout my entire graduate studies. Secondly, I would like to thank Dr. Xin Liu for his continuous guidance, discussion, and insightful feedback during my research work. This thesis would not have been possible without their guidance. Besides, I would like to thank members of Murata laboratory (former and present) for their encouragement, discussions, and constructive criticisms that helped me to achieve fruitful research outcomes. Regarding financial aid, I would like to acknowledge the financial support from the Japanese Government, Monbukagakusho (MEXT). Without their sponsorship, I would not have been able to pursue my graduate studies here at Tokyo Institute of Technology. Finally, I would like to thank my family for their support and encouragement during my graduate studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

In recent decades, computational and data storage technologies have advanced at a rapid rate. With these technologies, we now have the capability to store information about almost every aspect of human civilization. In addition to storing data related to our daily activities, we are also gathering data from tiny atoms, molecules, DNA, proteins, to large number of stars and galaxies.

With inquisitive human nature, we are always trying to understand the world around us and try to find patterns that could be useful. The vast amount of data available now provides us with a unique opportunity which has not been available before. However, the sheer magnitude of data poses a significant challenge in understanding and getting insights from the data. To approach this problem, the practitioner often categorizes the data and try to find appropriate tools that can help us to efficiently process and analyze the data. Some commonly used data formats are image format, text format, tabular format, audio format, etc. With the help of appropriate tools, it becomes easier to uncover hidden patterns and gain essential insights into the data. For example, by analyzing transaction data, financial institutions can detect and avoid fraudulent transactions. Similarly, email providers and social media companies can detect spam content or bots by taking advantage of the analysis of a significant amount of available text data.

Graphs are one of the important tools developed to represent the entities and relationships between them. The entities are represented by nodes, and the relationships between them are represented by edges. Often, edges exist between nodes that might share some common property or attributes. For example, in a social network, nodes represent the users, and edges represent the friendship between the

users. Users may share common attributes like geographical location, interests, political affiliation, workplace, etc. Graphs can be constructed on a wide-variety of data. They can represent social interactions [Mislove et al. 2007], communication systems [Monge, Contractor, and Contractor 2003], transport systems [Ferber et al. 2009], computer networks [Erciyes 2013], biological processes [Brohée and Helden 2006], and many more.

1.2 Why Graphs are important?

At first view, a graph may seem a simplistic representation of nodes connected by edges. However, graphs can be quite versatile. The edges can be used as such (undirected graphs) or can be assigned directions (directed graphs), emphasizing that the relationship is uni-directional. In addition, multiple edges or weighted edges can also exist between nodes. While creating graphs from the data derived from various sources, different connectivity patterns between nodes can be observed. Based on these connectivity patterns, graphs can often be categorized into different types. Random graphs, scale-free graphs, bipartite graphs, multi-layered graphs, etc. are some of these commonly used types of graphs.

The connectivity pattern of edges often encodes the information about real-world phenomenon associated with the dataset. For example, in scale-free graphs, few nodes are connected to most of the other nodes, while the majority of nodes have sparse connections with other nodes. Hence, the degree distribution often follows the power-law distribution. In real-world social networks, we often see that famous people/ influencers often have millions of followers. However, an average person has a few hundred to a few thousand followers. Hence, a set of a small number of nodes in social network graphs is significantly more influential in spreading information across the network. This knowledge is useful if we are interested in stopping the spread of misinformation or spreading awareness about a product among people.

Another example of using connectivity patterns in graphs is recommendation systems. Often users connected on social networks share common interests. Social media companies can then utilize this information to do selective targeting of advertisements on a group of people. Similarly, in e-commerce related websites, similar products can be connected in a graph, and if a user buys an item, similar items can be recommended, as they have a higher likelihood of being bought by the user.

1.3 Learning on Graphs with Neural Networks

Learning on graph-structured data has a long history, and researchers have been continuously improving the learning methods for better prediction capabilities. There are many approaches to solve graph learning problems [Xia et al. 2021]. Some of them are matrix factorization based methods [Kuang, Ding, and Park 2012], graph signal processing based methods [Ortega et al. 2018], random walk based methods [László 1996] and more recently deep learning based methods [Wu et al. 2019b].

Recently the field of deep learning has seen tremendous progress. The use of large-scale neural network models has continuously produced state-of-the-art (SOTA) results on a wide variety of real-world data and tasks. Naturally, researchers have also applied neural network models on graph-structured data and have achieved significant improvements over traditional approaches. These neural network models are commonly referred to as Graph Neural Networks (GNNs) [Wu et al. 2019b; Zhang, Cui, and Zhu 2020; Zhou et al. 2019]. The applications using GNNs range in wide variety of tasks like recommendation systems [Fan et al. 2019b; Ying et al. 2018a; Wu et al. 2021], graph data mining [Li et al. 2019; Ying et al. 2018b], molecular properties predictions/ molecule generation [Gilmer et al. 2017; Madhawa et al. 2019; Jin et al. 2019] , natural language processing [Yin et al. 2020; Guo, Zhang, and Lu 2019], vision tasks [Yang et al. 2018; Woo et al. 2018], physics simulations [Pfaff et al. 2021], node ranking [Maurya, Liu, and Murata 2019; Maurya, Liu, and Murata 2021; He et al. 2022] etc. Due to the robustness of GNNs, continuous effort is being made to improve the learning capabilities of GNN models by designing new architectures that can adapt to data with various underlying patterns and properties.

1.4 Inductive bias of Graph Neural Networks

Neural networks have been shown to learn on a wide variety of data. Often the architecture of the neural network is based on the type of input data. For example, image-based data uses convolutional neural networks, text data uses Recurrent Neural Networks (RNNs) or Gated Recurrent Units (GRUs), graph-structured data uses Graph Neural Networks, and so on. Recently, Transformers have evolved as general neural network models that can be adapted to vision tasks [Dosovitskiy et al. 2021], natural language processing (NLP) tasks [Vaswani et al. 2017] or a combination of both [Ramesh et al. 2021].

Most neural network model architectures are made of some common components. For example, almost all neural network models use neural network layers with learnable weights, non-linear activation like ReLU, Dropout for regularization, etc. The

architecture of the models is often designed based on the type of input and task at hand. This is often considered as the inductive bias in the design of neural network models. For image/video related data, convolution operation is necessary and is useful to learn translation-equivariant features [Mouton, Myburgh, and Davel 2020]. Even recent vision transformer models require patches of images as input which are equivalent to the convolution operation on the images [Dosovitskiy et al. 2021]. Similarly, with text data, neural networks learn to preserve the context of words with other words in a sentence. Different types of NLP models like RNNs, LSTMs, GRUs, Transformers models, etc, use this inductive bias while training on the text data. Even though deep learning models are evolving and becoming more sophisticated, the fundamental inductive biases of these models remain the same. Researchers have proposed numerous ways to modify the architectures to make them more complex and expressive, thus enhancing their learning capabilities. For example, the use of residual layers has helped to make models deeper, or the use of attention layers helps to emphasize useful features during the model training.

Graph neural networks use feature aggregation operation based on the graph structure as the inductive bias to compute node features. Features from the neighboring nodes are aggregated over multiple hop distances from the source node, and final node embeddings are computed, which can be used in downstream tasks. Hence the aggregation operation plays a significant role in learning via graph neural networks.

1.5 Importance of Feature Aggregation

As the graph structure can store a wide variety of information, learning on a graph is a very complex problem and is highly dependent on the type of downstream tasks. For example, in social network, we might be interested in finding local clusters of people sharing some common attribute; in contact networks, we might be interested in how disease might spread among people; in transport networks, we might be interested in finding optimal paths to reduce time/cost in deliveries; in drug discovery research, we model molecules as graphs and train models to generate new molecule structures with desirable properties, and so on.

The aggregation scheme in a GNN model dictates the feature propagation and aggregation step. It helps to learn and map the structure of the graph and node/edge features into meaningful and useful node embeddings. In the last few years, there have been many variants of GNN models proposed to improve the learning aspect of the GNNs. Some of these models have improved the neural network component of GNNs, like the introduction of residual layers, transformer layers, etc. Some other

models have proposed to improve the feature aggregation step, like the introduction of the random sampling of neighbors to improve scalability, using attention layers to assign weights to node's neighbors for feature aggregation step.

As the GNNs are now being applied to many different fields, the same aggregation scheme might not be helpful for every task. It is imperative to modify and improve the default aggregation scheme in congruence with the task. Recently, researchers have proposed GNN models for specific tasks that leverage domain knowledge and use modified aggregation schemes that can perform better than GNN models with the simple aggregation scheme. Some of these models are PC-GNN [Liu et al. 2021], PAGTN [Chen, Barzilay, and Jaakkola 2019], GMT [Baek, Kang, and Hwang 2021], etc.

1.6 Thesis Synopsis

1.6.1 Motivation

Recent years have seen rapid growth of Graph Neural Networks (GNNs) being utilized in real-world applications in a multitude of domains. These applications range in wide variety of tasks like node classification [Kipf and Welling 2017; Velickovic et al. 2018b; Abu-El-Haija et al. 2019; Chen et al. 2020; Wang and Derr 2021], link prediction [Ying et al. 2018a; Berg, Kipf, and Welling 2017; Chami et al. 2019], graph classification [Ying et al. 2018b; Zhang et al. 2018], prediction of molecular properties [Gilmer et al. 2017; Madhawa et al. 2019], node ranking [He et al. 2022; Fan et al. 2019a] and natural language processing [Marcheggiani and Titov 2017].

However, we observe that many of these applications use basic GNN architectures with default aggregation scheme similar to earlier GNN models like GCN [Kipf and Welling 2017]. The default aggregation scheme, while useful, may not be optimal for all applications, as they may have different requirements related to graph learning.

Hence, there is a need to explore application-oriented GNN feature aggregation schemes. Since introduction of earlier GNN models [Kipf and Welling 2017; Defferrard, Bresson, and Vandergheynst 2016], researchers have proposed variety of improvements in the GNN models. Some of the techniques used in these variants include neighbor sampling [Hamilton, Ying, and Leskovec 2017; Chen, Ma, and Xiao 2018], attention mechanism [Velickovic et al. 2018b], use of Personalized PageRank matrix instead of adjacency matrix [Klicpera, Bojchevski, and Günnemann 2018], leveraging proximity in feature space [Jin et al. 2021] and simplified model design [Wu et al. 2019a]. Also, similar to deep vision models, there have been proposals to make GNN models deep by stacking a large number of layers and using residual

connections to stabilize the training [Rong et al. 2020; Chen et al. 2020]. Most of these works have approached the problem by bringing in architecture design and model training techniques from the vision and natural language processing (NLP) fields, and they have indeed improved the performance of the GNN models. However, we still need to develop approaches from graph-centric point of view. Since feature aggregation in GNNs is a fundamental and graph-centric operation, in this work, we explore how it can be adapted and improved based on the given downstream task.

In this thesis, we explore two different problems: node ranking and node classification. Node ranking is the problem of identifying the importance of the nodes in terms of connectivity and information propagation based on the structure of the graph. For our work, we choose two ranking measures: Betweenness Centrality and Closeness Centrality. These node ranking measures are based on shortest-path distances in a graph and are computationally expensive to calculate. There are many node centrality approximation methods in the literature; however, they suffer from the trade-off of ranking accuracy versus execution time. We aim to use the GNN model to learn to predict node rankings of the input graph and take advantage of its low inference time. We propose a new aggregation scheme that is based on the approximation of the shortest paths in a given graph. Using it, we propose two GNN models [Maurya, Liu, and Murata 2019; Maurya, Liu, and Murata 2021], GNN-Bet and GNN-Close to approximate betweenness and closeness centrality, respectively. In addition, we experimentally found that our proposed aggregation scheme is better than the default aggregation scheme of GNN models.

For the node classification task, in a simple GNN model, features of neighboring nodes are aggregated over all hops as set by the hyper-parameter. However, we experimentally show that this scheme is not useful in case node features show *heterophily* characteristics in the graph structure and can cause the adverse performance of the model. In addition, features of neighboring nodes located at multiple hops may not be similar to the source node and act as noisy feature information for the source node. We experimentally verify the degradation of the model's performance, in this case, on the node classification task. Hence we propose an adaptive aggregation scheme that learns to select features from hops that are useful and ignores others. Our proposed new GNN model [Maurya, Liu, and Murata 2022b; Maurya, Liu, and Murata 2022a] outperforms other state-of-the-art GNN models on real-world node classification datasets.

In both our works, we improve the GNN models by proposing modifications of the aggregation scheme over the default aggregation scheme of GNN. These modifications are inspired by the knowledge derived from the task of node ranking and node classification. For the node ranking model, we propose the aggregation scheme

that aggregates feature information along possible shortest paths in the graph. For the node classification model, we propose the aggregation scheme that dynamically learns to select the best possible hop-features with respect to the label distribution of the nodes. Our work follows the recent trend in GNN literature where new GNN models are proposed that utilize domain/task-specific knowledge in their architecture design. Our work or other recent works in GNN literature show that the traditional design of the GNN model, like GCN, often fails to perform optimally on new different tasks. With the improvements in the GNN architecture, new GNN models outperform traditional GNN models and are more useful.

1.6.2 Structure and Organization

The thesis is organized into the following structure:

1. In Chapter 1, we present a brief introduction to graphs, why their study is essential, and how graph neural networks have an important tool for graph learning on real-world data. We discuss the importance of inductive bias in the design of graph neural networks and neural networks in general. Taking it into consideration, we discuss the importance of feature aggregation schemes and provide brief explanation of our proposed GNN models for the tasks of node ranking and node classification.

2. Chapter 2 provides an overview of terminologies and concepts which have been used in this thesis. We discuss the basic properties of the graph, introduce neural networks, and provide details on graph neural networks and feature aggregation operation.

3. Chapter 3 provides detailed information on different aggregation schemes in GNN literature. We discuss the distinctions between general-purpose and domain-specific aggregation schemes and the current trend of new GNN models with improved aggregation schemes.

4. In Chapter 4, we introduce two centrality measures: Betweenness Centrality and Closeness Centrality. We discuss challenges in calculating these measures for large graphs and the shortcomings of approximations methods. We then propose our GNN model, which learns to approximate these measures by using a modified feature aggregation scheme based on the shortest paths in the graphs. We conduct extensive experiments on synthetic and real-world graph datasets and report results of ranking performance on all nodes and top-k nodes.

5. In Chapter 5, we introduce the problem of node classification in graph learning. We introduce concepts of Homophily and Heterophily in graph-structured data and demonstrate that a simple aggregation scheme is not optimal for learning. We present our proposed model that learns to dynamically adapt aggregation scheme over multiple hops and leads to improvements in node classification on real-world datasets.

6. In Chapter 6, we discuss the design considerations, limitations, and applications of our proposed GNN models. We then present a unified framework of feature aggregation that combines both our works and paves the way to improve GNNs by modifying feature aggregation schemes in GNNs.

7. Chapter 7 provides key contributions and the future directions of our work.

In any research field, the reproducibility of research and related experiments are of paramount importance. Hence, we aim to provide comprehensive information about our experiments including model design details, hyper-parameters, software environment and hardware details. In Appendix A, we provide methods, statistics, and hyper-parameters of generating synthetic graphs used in experiments in Chapter 4. In Appendix B, we provide training hyper-parameters for our proposed model FSGNN. We also include additional experiments to show the performance of our model under more varied settings and the scalability of the model on large datasets.

1.7 List of Publications

Here, we provide a list of publications that this thesis is centered on.

Journals

- **Sunil Kumar Maurya**, Xin Liu, Tsuyoshi Murata. Graph Neural Networks for Fast Node Ranking Approximation. In *ACM Transactions on Knowledge Discovery from Data (TKDD)*. May, 2021.
- **Sunil Kumar Maurya**, Xin Liu, Tsuyoshi Murata. Simplifying Approach to Node Classification in Graph Neural Networks. In *Journal of Computational Science*. July, 2022.
- **Sunil Kumar Maurya**, Xin Liu, Tsuyoshi Murata. Feature Selection: Key to Enhance Node Classification with Graph Neural Networks. In *CAAI Transactions on Intelligence Technology*. January, 2023.

Conferences

- **Sunil Kumar Maurya**, Xin Liu, Tsuyoshi Murata. Fast approximations of betweenness centrality with Graph Neural Networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*. November, 2019.
- **Sunil Kumar Maurya**, Xin Liu, Tsuyoshi Murata. Not All Neighbors are Friendly: Learning to Choose Hop Features to Improve Node Classification. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM)*. October, 2022.

Chapter 2

Background

In this chapter, we briefly introduce the topics that will build the foundation of our discussion in later chapters. Topics specifically related to the graph-related tasks in Chapter 4 & 5 are introduced and discussed in relevant chapters.

2.1 Notations

Let $G = (V, E)$ be an undirected graph with n nodes and m edges. For numerical calculations, graph is represented as adjacency matrix denoted by $A \in \{0, 1\}^{n \times n}$ with each element $A_{ij} = 1$ if there exists an edge between node v_i and v_j, otherwise $A_{ij} = 0$. For undirected graphs, their adjacency matrices are symmetric, i.e. $A_{ij} = A_{ji}$. When self-loops are added to the graph then, resultant adajcency matrix is denoted as $\tilde{A} = A + I$. Diagonal degree matrix of A and $\tilde{A}$ are denoted as D and $\tilde{D}$ respectively. Usually each node is associated with a d-dimensional feature vector and the feature matrix for all nodes is represented as $X \in \mathbb{R}^{n \times d}$. In our discussion, we will interchangeably refer to feature matrix as features of the nodes.

2.2 Properties of Graphs

In any type of graph learning task, graph structure plays a crucial role and often captures the properties of underlying processes that generate the data. Contact networks/ social networks often show local clusters where edge density among the nodes is higher than average and represent community among people in real life. Road networks often form grid-like graph structures. Citation networks are represented both as directed or undirected graphs. Directed citation graphs can be used to tempo-

rally infer which papers were published earlier than others. With these examples, it is easy to observe that graph-structured data can vary widely and often require targeted approaches to solve a given problem.

As the literature on graphs is vast, it is difficult to summarize all properties of graphs here. In the following section, we describe graph-related terminologies that we will be using in chapters ahead.

2.2.1 Paths

A path in a graph is a sequence of nodes such that every consecutive pair of nodes in the sequence is connected by an edge in the graph structure [Newman 2010]. Paths can be defined for both directed or undirected graphs. For directed graphs, the path must be traversed along the direction of the edges. Hence it is possible for a path to exist between two nodes in the undirected graph but not in the directed graph. Often, paths in a graph are calculated to analyze how information flows from one node to another.

Shortest Path

The shortest path between two nodes is the path that traverses the least number of edges. It is possible to have many shortest paths (same length) between two nodes.

2.2.2 Distance

The distance between two nodes is the length of the shortest path between two nodes. In GNN literature, the term *hop* is often used to denote how far the node is from the source node.

2.2.3 Centrality Measures

In any given graph derived from the data, it is helpful to know which nodes are more important than the others. For example, in a social network, a person with a large following (high degree) is more useful for spreading information across network than a person with a small number of followers (low degree). In network science, many measures have been developed to calculate the importance of nodes given a certain criterion. These measures are collectively called as centrality measures. Some frequently used centrality measures are Betweenness Centrality, Closeness Centrality, Degree Centrality, Eigenvector Centrality, PageRank etc. Often these centrality measures assign a numeric value to each node called as a centrality value. A node

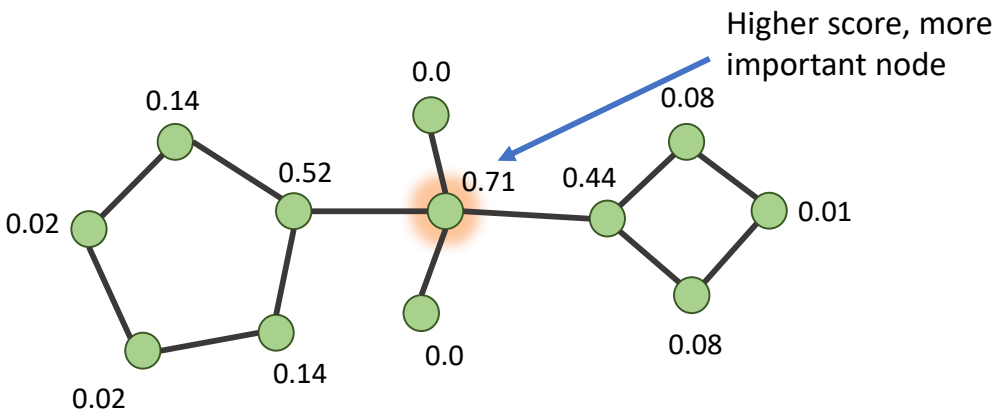


Figure 2.1: In general, higher importance nodes are assigned higher centrality scores

with a higher centrality value than other nodes will be denoted as a high-ranking node for the centrality measure (Figure 2.1).

2.2.4 Node/Edge Features

Graph structure from any real-world data is often derived after pre-processing of data. In addition to the graph structure, data may contain numeric or categorical values describing attributes of the nodes or edges. This attribute information is referred to as node/edge features in machine learning literature. Taking the example of a social network, nodes and edges define users and their friendship status with other users. Each user also has attributes like gender, location, age, political affiliation, interests, etc. These attributes can be encoded numerically and considered as node features.

2.3 Neural Networks

In the last decade, neural networks have rapidly become one of the go-to tools for researchers on a wide variety of problems. Neural network models have achieved state-of-the-art (SOTA) performance on various types of tasks and data. Though they have become successful recently, they were envisioned in the 1940s and 50s. They achieved significant milestones with the invention of the backpropagation al-

gorithm and the introduction of CNNs. In the late 2000s and early 2010s, due to availability of sufficient computing resources and large quantities of data, it became easier to train neural networks in real-world applications. Since then, neural networks have improved due to better architecture design, training methodology, and availability of ever-increasing data.

Basic block of neural network is defined as a Perceptron. For given input signals $x_1, x_2, x_3, \cdots, x_m$, learnable weights $w_{k1}, w_{k2}, w_{k3}, \cdots, w_{km}$, o_k is the linear combination of input signals.

$$o_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1}$$

Scalar value of bias b_k is added to o_k which has the effect of applying affine transformation. To introduce non-linearity, $\phi(\cdot)$ activation function (e.g. sigmoid) is used. Combining these, we get neuron functionality which is defined as,

$$y_k = \phi(o_k + b_k) \tag{2.2}$$

In practice, many layers of neural network layers are stacked together to create Deep Neural Networks. Deep neural networks have the ability to learn complex patterns present in the data that are useful for downstream tasks. In addition, many other techniques like the use of different types of non-linear activation functions like ReLU, GeLU, LeakyReLU, etc., use of regularization techniques like Dropout and weight decay, and splitting data into smaller batches are used to improve the quality and efficiency of the training. During training, the output of the neural network is compared with the desired target value, an error is calculated, and using the backpropagation algorithm, gradients with respect to the neural network weights are calculated. Using optimizers like Adam, AdamW, AdaGrad etc. weights are updated and next training iteration is continued. Training is continued till a criterion like early stopping is fulfilled, and the trained neural network is deployed on test data or data in production.

2.4 Graph Neural Networks

With the rapid advancement in technologies, we now have access to vast troves of data. A significant portion of this data is non-euclidean or irregular by nature. Hence, graphs become the default tool to process this type of data. Analysis of graph-structured data can often uncover hidden attributes or missing information. Graph Neural Networks (GNNs) are neural network models commonly used

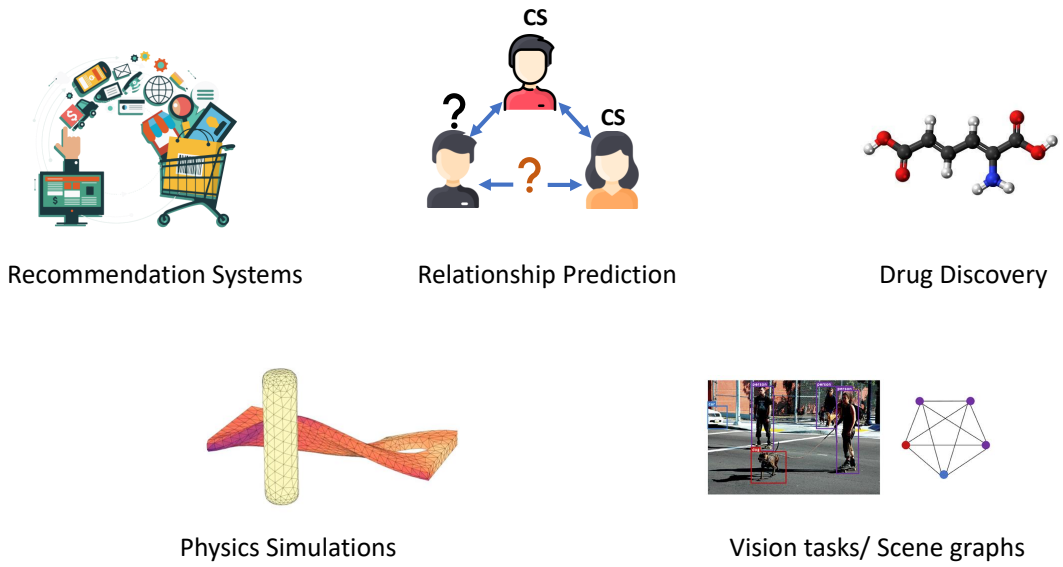


Figure 2.2: Applications of GNNs

on graph-structured data. Based on user-defined relationships in the data, a graph is constructed, and nodes/edges are assigned attributes (or features) in the form of real-valued vectors. Using these feature vectors as input, GNNs perform feature aggregation step where each node collects feature information from its neighbors based on the graph. Utilizing the feature aggregation step, GNNs learn over the similarity/dissimilarity of node features and use this information for various prediction tasks. With rapid development of the field and easy accessibility of computation and data, GNNs have been used to solve a variety of problems like node classification [Kipf and Welling 2017; Velickovic et al. 2018b; Abu-El-Haija et al. 2019; Chen et al. 2020; Wang and Derr 2021], link prediction [Ying et al. 2018a; Berg, Kipf, and Welling 2017; Chami et al. 2019], graph classification [Ying et al. 2018b; Zhang et al. 2018], prediction of molecular properties [Gilmer et al. 2017; Madhawa et al. 2019], node ranking [Maurya, Liu, and Murata 2021; Fan et al. 2019a] and natural language processing [Marcheggiani and Titov 2017].

For a GNN model, input is the graph structure, usually in the form of an adjacency matrix and node/edge feature matrix. Feature aggregation step is performed implicitly in the GNN layer, or explicitly pre-computed, and the neural network is trained to predict node property value as close as possible to the actual node property value (target value). Figure 2.3 shows a simple formulation of a GNN model.

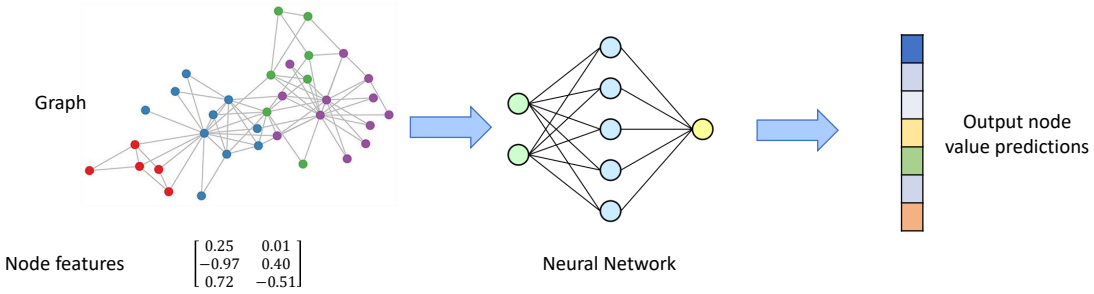


Figure 2.3: Simple Formulation of Graph Neural Network. Input is graph structure and node (/edge) features. Output is predicted node property which is compared with the target value.

Graph neural networks are often used for tasks like classification, regression, generative modeling, etc. The output of the GNN model is designed based on the given task in hand. For node classification tasks, the model predicts probabilities of label assignment. In molecular drug design tasks, the model is designed to predict the properties of molecules represented as graphs using regression-based learning. Also, the GNN model may be trained to learn to generate new molecule structures following some statistical properties learned from the data in the training set.

Hence the study of the graph neural models is a very vast, diverse, and challenging field. Due to its immense benefits and applicability in multiple domains, there has been a significant focus and continuous effort from researchers to improve the capabilities of GNN models.

2.5 Feature Aggregation

Feature Aggregation is the fundamental operation of a GNN model. In each aggregation step, nodes aggregate the features of their neighboring nodes (Figure 2.4). The number of aggregation steps for a model is set as a hyperparameter of the model training. In a default aggregation step, nodes are explored in a Breath-first search (BFS) type setting, and features from all nodes are aggregated by the source node. Some works [Chen, Ma, and Xiao 2018; Hamilton, Ying, and Leskovec 2017] have proposed to instead randomly sample nodes from a set of all explored nodes and combine the features of those sampled nodes. This step was proposed to scale GNN models to large graphs with millions of nodes.

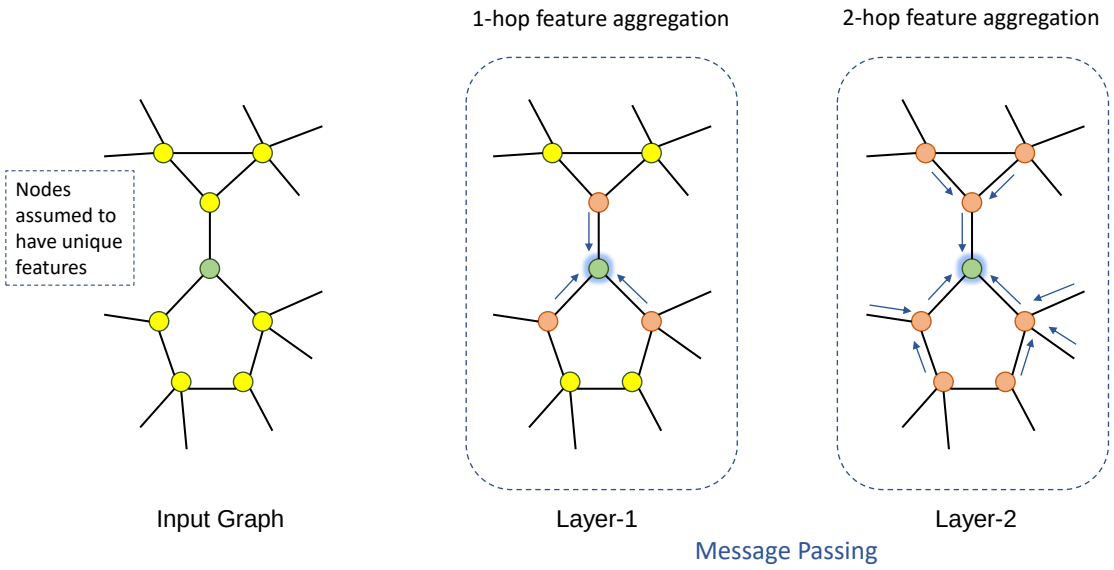


Figure 2.4: Feature aggregation over two hops

However, the default aggregation scheme, while useful, is not optimal for all types of tasks. For example, on graph learning tasks that rely on shortest path information flow or tasks on graphs where nodes with dissimilar features are connected, default aggregation might not be suitable and, in the worst case, can degrade the performance of the GNN model. Hence, it is necessary to explore new ways to perform feature aggregation relevant to the task. In recent years, many works have proposed improved feature aggregation schemes that are relevant to domain-specific tasks [Liu et al. 2021; Chen, Barzilay, and Jaakkola 2019; Baek, Kang, and Hwang 2021]. These improvements often entails including domain knowledge to modify the default aggregation scheme.

Chapter 3

Aggregation Schemes in Graph Neural Networks

3.1 Introduction

Data from various domains can be modeled as a graph with entities represented as nodes and relationships between them as edges. Graph Neural Networks leverage graph structure to utilize the relationships between the nodes. These edges can provide additional information or context that can help to improve learning on given data and provide better predictions. For example, in a social network, users with similar attributes might be connected with each other. These attributes can be work location, interests, political affiliations, etc. In many instances, a user may have incomplete information on their profile regarding these attributes. In those cases, we may use the graph structure of the social network to learn and predict these missing attributes (Figure 3.1).

3.1.1 Default Aggregation Scheme

In a default aggregation scheme, a node combines the feature information from all its neighbors. If we repeat this process, the node can get the feature information of neighbors of neighbors, thus effectively increasing the scope of information gathering. The node explores its neighborhood similar to the Breadth-first search (BFS) type setting. The range of neighborhood exploration is determined by the number of times the feature aggregation step is applied and is set as a hyper-parameter of the GNN model. The feature aggregation step, in general, is implemented as a multiplication of the adjacency matrix with the node feature matrix, i.e., AX, where A denotes

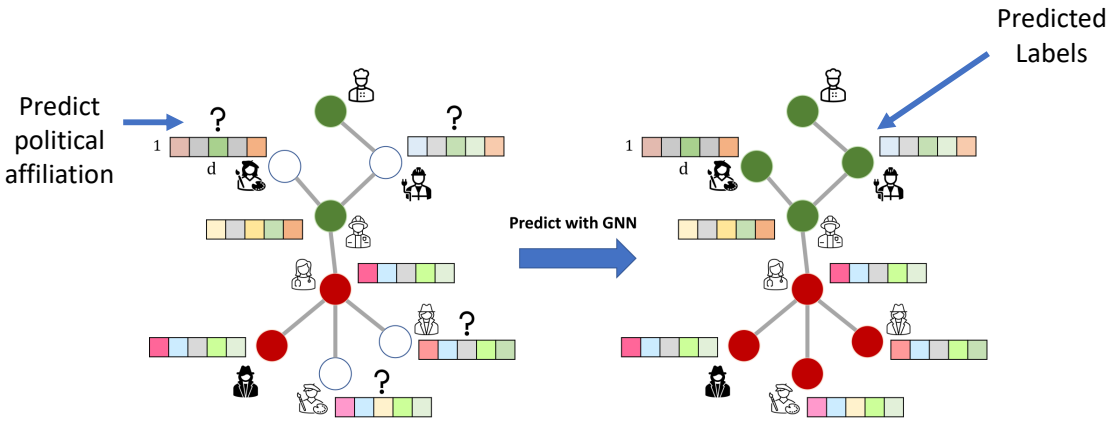


Figure 3.1: Users with similar attributes are likely to be connected with each other. Using graph structure with users as nodes and friendship status as edges, missing attribute information can be predicted for users.

the adjacency matrix and X denotes the node feature matrix. k-step aggregation is implemented with the k-th power of the adjacency matrix as $A^k X$. The feature aggregation step can be either embedded in the GNN layer or aggregated features can be pre-computed and then can be used as input to a neural network model. These two formulations are described as follows:

$$H^{(i+1)} = \sigma(AH^{(i)}W^{(i)}) \tag{3.1}$$

Equation (3.1) denotes the formulation of feature aggregation embedded in the GNN layer. $H^{(i)}$ denotes node features/embedding input to i^{th} layer, $W^{(i)}$ is the learnable weight matrix in the neural network layerand σ denotes the non-linearity like ReLU. Multiple GNN layers can be stacked together, and output is used for the downstream task. This formulation is utilized by many GNN models [Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017; Velickovic et al. 2018b; Chen et al. 2020].

$$Y = \text{softmax}(A^k X\Theta) \tag{3.2}$$

Equation (3.2) denotes the pre-computation of aggregated features, and the resultant matrix is mapped to the classifier Θ. Softmax function converts the output logits of the classifier to the relevant probability values with respect to the class

labels. This formulation was proposed in the SGC model [Wu et al. 2019a] and has been used in other GNN models [Frasca et al. 2020; Maurya, Liu, and Murata 2022b].

3.1.2 Problems associated with GNN aggregation scheme

In GNN literature, researchers have proposed numerous variants of GNN models applicable for a wide range of tasks like node classification, graph classification, graph regression, link prediction, etc. In the process of studying GNN models, researchers have identified two problems described as follows:

Oversmoothing in GNNs

In the default aggregation scheme, the number of neighboring nodes increases as we increase the aggregation steps. With a high number of aggregation steps, the aggregated features of the nodes converge and tend to become similar to each other. This creates a problem of distinguishing embeddings of different nodes and causes the performance of the model to drop [Chen et al. 2019; Oono and Suzuki 2020]. Due to this phenomenon, many conventional GNN models cannot be designed with an arbitrarily high number of GNN layers. Recently, many architectural improvements have been proposed in GNN literature that helps to alleviate the oversmoothing problem and enable a practitioner to create deep GNN models [Chen et al. 2020; Li et al. 2021; Rong et al. 2020].

Oversquashing in GNNs

The oversmoothing problem in GNNs is a widely known phenomenon, and many publications in GNN literature have provided in-depth analysis and ways to alleviate this problem. Recently, researchers have brought into light another problem associated with GNNs, which is known as the oversquashing problem [Alon and Yahav 2021; Banerjee et al. 2022]. With each subsequent aggregation step, the number of neighboring nodes can increase exponentially. However, the aggregated feature information is encoded into fixed-dimension embedding. This leads to a substantial loss of information. [Alon and Yahav 2021] highlight that this problem is more prominent in GNN models, which treat all neighbors as equal like GCN [Kipf and Welling 2017] and GIN [Xu et al. 2019]. While GNN models with weighted aggregation schemes like GAT [Velickovic et al. 2018b] and GGNN [Li et al. 2017b] are more effective against this problem. This research area is still under exploration, and more results and solutions tackling this problem might come in the future.

3.2 Modified Aggregation Schemes

As discussed in Section 3.1, the feature aggregation step helps to encode graph structure into the node embeddings. Though many GNN models use the default aggregation scheme, many researchers in recent years have proposed modified versions of the aggregation scheme. These modifications are either based on some constraints or inspired by the domain knowledge of the given task. We separate the modified aggregation schemes in two categories. The first category includes general-purpose schemes that can be employed in any given task. The second category includes aggregation schemes that can be applied to models learning on tasks related to a specific domain.

3.2.1 General-Purpose Aggregation Schemes

The modified aggregation schemes in this category are proposed to solve general problems encountered while training GNN models. Some of the objectives of these schemes are scaling GNN models to very large datasets, adding regularization in model training to avoid over-fitting and improved robustness, improving the learning capability of the GNN model, etc.

Scaling GNN models

These schemes propose sampling of neighbors for feature aggregation instead of using features of all neighbors of the node. By fixing the sample size, the GNN model can be trained in small batches on large datasets with millions or billons of nodes. FastGCN [Chen, Ma, and Xiao 2018], GraphSAGE [Hamilton, Ying, and Leskovec 2017], and PinSage [Ying et al. 2018a] are examples of GNN models that employ sampling-based feature aggregation schemes. One potential disadvantage of this approach is loss of the feature information due to neighborhood sampling, which can cause a reduction in the performance of the GNN model.

Improving learning capability of GNNs

Much of deep learning progress has been achieved by the improvements in architectural designs and the addition of novel neural network components like attention layers, residual layers, and so on. For GNNs, GAT [Velickovic et al. 2018b] introduced the attention mechanism on edges to weight the features aggregated from the neighbors. GATv2 [Brody, Alon, and Yahav 2022] improves the implementation of GAT by introducing the concept of dynamic attention. [Yun et al. 2020] introduces

transformer architecture in GNNs that improves over the default aggregation scheme by learning meta paths in heterogeneous graphs. H2GCN [Zhu et al. 2020] introduces the concept of two types of aggregation, with self-loop and without self-loop aggregation. Though the motivation behind this idea is homophily and heterophily in social graphs, this aggregation scheme can be implemented and used in any domain.

Adding regularization in GNN training

Regularization has become an essential component of training neural networks. Dropout and weight decay are the two most commonly used regularization techniques. DropEdge [Rong et al. 2020], and GRAND [Feng et al. 2020] introduces the concept of randomly dropping edges or nodes during training to force the model to learn more robust embeddings. However, these techniques can be considered as a partial modification of the aggregation scheme as they are only applied during training of the GNN model, and during inference, the complete graph is present in the input.

3.2.2 Domain-specific Aggregation Schemes

In the previous section, we discussed modified aggregation schemes that, in general, can be applied to problems from any domain and do not account for any domain knowledge that might help to improve the performance of the GNN model. However, as the GNN models are now being used in a wide range of fields, leveraging domain knowledge can provide better prediction capabilities. In this section, we discuss some new feature aggregation schemes from GNN literature that improve upon default aggregation scheme with domain-specific bias.

Need for new aggregation schemes

In the default aggregation scheme, all neighbors are considered equal, and graph structure or node information does not affect the importance of nodes in the feature aggregation step. However, a real-world data often exhibit wide-range of properties; e.g., the connected nodes may be of different types or have an imbalanced ratio, the graph structure and connectivity can range from homogeneous to power-law distribution, missing feature/label information, etc. These properties often depend on domain characteristics or the method by which the data is collected. These factors require a more customized approach to the design of the GNN model. The following are some examples where researchers have proposed new task-oriented GNN models.

1. **PC-GNN** Pick and Choose Graph Neural Network (PC-GNN) [Liu et al. 2021] solves the problem of detecting fraud detection in graph-structured data.

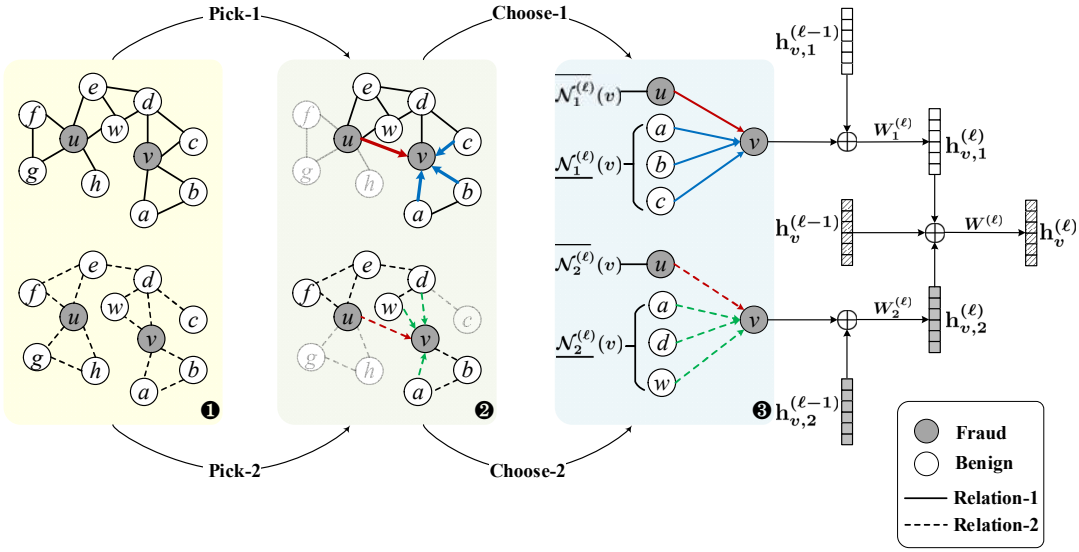


Figure 3.2: Aggregation Scheme of PC-GNN model. Nodes are sampled to maintain label-balance of fraudulent vs non-fraudulent nodes.

Fraud detection is challenging as the number of fraudulent nodes are very small compared to normal nodes. While training a GNN model on the data, using a simple aggregation scheme would lead to the dilution of feature information of fraudulent nodes due to their small number resulting in the model's failure to detect such nodes. PC-GNN proposes a label-balanced sampler to bring back the balance of normal vs. fraudulent nodes, which leads to better fraud detection capabilities of the model.

2. **PAGTN** This model is proposed for the prediction of molecular properties. Authors of the paper [Chen, Barzilay, and Jaakkola 2019] state that default aggregation scheme in models like GCN exploit the feature locality of graphs and long-range dependencies are difficult to learn. PAGTN model aggregates feature information from all nodes by utilizing path features in global attention layers, resulting in a richer and more expressive model. Path features between each node pair are computed by using the shortest path between them. The experimental results show the PAGTN model performs better than GCN. Similar to PAGTN, SPN model [Abboud, Dimitrov, and Ceylan 2022] also proposes a shortest-path based message passing scheme for GNNs to improve graph property prediction. The authors also show that shortest-paths networks alleviate

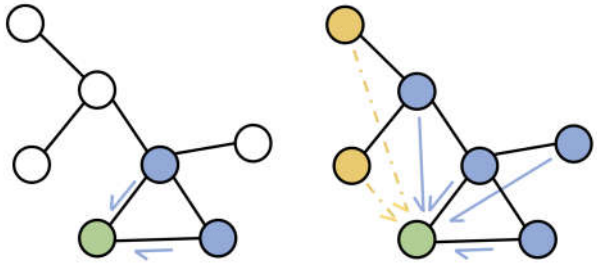


Figure 3.3: PAGTN model proposes to aggregate the feature information from all nodes in the graph via the shortest paths between node pairs.

the problem of over-squashing in GNNs.

3. **GMT** Graph-level tasks require a two-step aggregation process. First, the features of neighbors of nodes are aggregated. Then to get graph-level embeddings, features of all nodes are combined using a pooling operation which can be sum-pooling, mean-pooling, or max-pooling. The authors of the GMT paper [Baek, Kang, and Hwang 2021] state that simple default pooling operation cannot generate meaningful graph representation in a task-specific manner. GMT encodes the given set of node embeddings as multisets and considers both the global structure of the graph and their task relevance in compressing the node features.

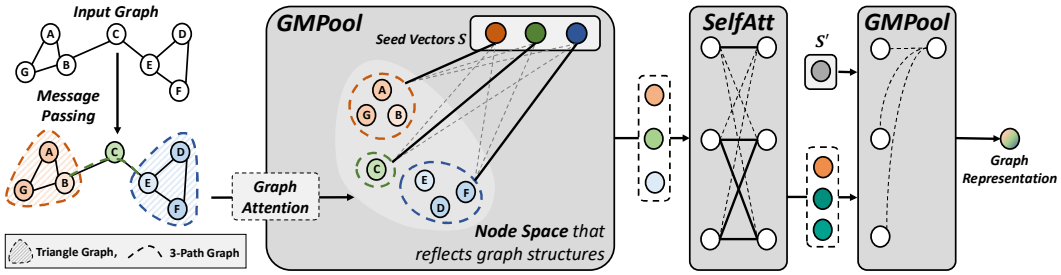


Figure 3.4: GMT model improves over the default pooling scheme by learning to group nodes into multisets using an attention-based pooling block (GMPool) and a self-attention block (SelfAtt) to compress the nodes into a few important nodes.

3.3 Discussion on Aggregation Schemes

In previous sections, we explored different types of aggregation schemes along with their utilities for different purposes. Based on the discussion earlier, we can broadly classify these schemes into two categories: General-purpose schemes and domain/task-based schemes.

In Table 3.1, we categorize various GNN models broadly into these two categories. Many GNN models like GCN [Kipf and Welling 2017], GraphSAGE [Hamilton, Ying, and Leskovec 2017], GIN [Xu et al. 2019], FastGCN [Chen, Ma, and Xiao 2018], etc. are categorized into general-purpose category. While models like PC-GNN [Liu et al. 2021], PAGTN [Chen, Barzilay, and Jaakkola 2019], SPN [Abboud, Dimitrov, and Ceylan 2022], etc. are designed and proposed based on domain-knowledge and to tackle certain challenges in the specific field. Hence, they are categorized into domain/task-specific category.

Table 3.1: GNN models are broadly divided into two categories based on the aggregation schemes. Underlined models are proposed and discussed in this thesis.

General-Purpose Scheme		**Domain/Task Specific Scheme**	
GNN Model	Objective	GNN Model	Objective
GCN, GIN	Simple aggregation	PC-GNN	Minority class prediction
GraphSAGE, FastGCN	Scaling GNNs	PAGTN, SPN	Molecular property prediction
DropEdge, GRAND	Regularization in training	GNN-Bet, GNN-Close	Node Ranking
GAT, GATv2	Attention layer in GNNs	FSGNN	Feature Selection in GNN

Need for introducing task-based aggregation schemes

As discussed in Sections 3.1 and 3.2, the default aggregation scheme is not always optimal for tasks and also exhibit some problems like oversmoothing and oversquashing. As GNNs are now being applied to a wide range of fields, it becomes an interesting and challenging problem to adapt the aggregation scheme to the given task to get the most out of GNN-based learning. As discussed in the previous section, researchers are proposing new GNN models by improving the aggregation schemes that take advantage of the domain knowledge and show better learning capability over traditional GNN models. In this thesis, we tackle the problem of node ranking and node classification and propose new GNN models with better aggregation schemes providing better performance.

Chapter 4

Node Ranking

4.1 Introduction

Graphs provide a simple way to abstract the data into simple structures that are easy to understand and convenient to process. They preserve the flow of information and interaction patterns in real-world data. Thus graphs can be used to uncover properties of the data and gain meaningful knowledge from it. One of the crucial tasks is to find the ranking of the nodes in graphs in terms of information spread and connectivity[Liao et al. 2017; Medo 2013]. Node ranking helps narrow down our focus from a huge population of data points to small and more relevant data points. Based on the situation, it can help us to take appropriate actions. For example, in the case of infectious diseases[Wang et al. 2016; Brockmann and Helbing 2013], it can help to find susceptible people who are more prone to spread disease once infected. Targeted immunization can curb the disease spread in its infancy and before it becomes an epidemic while minimizing resource utilization. Similarly, in social networks, spread of misinformation can be prevented[Pei et al. 2014]. In road and power infrastructures, finding critical points can help to take preemptive actions to make systems more robust to failures. Thus the study of node ranking is an essential part of graph analysis.

In general, researchers have proposed centrality measures for node ranking and these measures provide us a quantitative view of how nodes play a certain role in the graph and how their presence or absence affects the graph structure as a whole. Some of the proposed measures are Betweenness Centrality[Freeman 1977; Brandes 2001; Newman 2005], Closeness Centrality[Bavelas 1950; Sabidussi 1966; Borgatti 1995], PageRank Centrality[Brin and Page 1998; Ding et al. 2009; Franceschet 2011; Gleich 2015] and so on. All of these measures rank nodes based on certain criteria. In this

paper, we will focus on two widely used shortest path based measures: Betweenness Centrality and Closeness Centrality.

Betweenness centrality ranks nodes based on how they facilitate the information flow in the graph. Removal of nodes with high betweenness can disrupt the flow of information in graph structure significantly[Borgatti 1995]. Betweenness centrality has been used for various purposes such as analysis of knowledge networks[Pham and Klamma 2010], contingency analysis in power grid systems[Jin et al. 2010], the study of biological graphs[Joy et al. 2005] and traffic monitoring in transportation networks[Puzis et al. 2012].

Closeness centrality is a measure of how close a given node is with respect to all other nodes in the graph. It determines the ease with which a node can propagate information across the graph. Closeness centrality has been used to identify influential nodes in social network community[Mika 2004], and to determine the expected arrival or hitting time for information or disease spreading through the community[Borgatti 1995; Borgatti 2005].

Both betweenness centrality and closeness centrality require the calculation of the shortest paths in the graph. As such, calculating the exact centrality value is prohibitive for large graphs in practice, many approximation algorithms based on sampling techniques have been proposed. Geisberger et al.[Geisberger, Sanders, and Schultes 2008] propose a method to approximate betweenness centrality by sampling k starting nodes (pivots) to find the shortest paths and estimate betweenness centrality for all nodes. Riondato et al.[Riondato and Kornaropoulos 2016] take a different approach while providing a theoretical guarantee. r shortest paths are chosen between randomly sampled source-target node pairs, and betweenness is approximated based on these paths. Borassi's algorithm [Borassi and Natale 2019] provides an adaptive sampling technique for sampling shortest paths, which provides a faster computation of betweenness within a given absolute error. For approximations of closeness centrality, Eppstein et al.[Eppstein and Wang 2001] and Okamoto et al.[Okamoto, Chen, and Li 2008] provide algorithms based on single-source shortest path computations on uniformly sampled nodes for a given graph. Cohen et al.[Cohen et al. 2014] propose an improved algorithm based on a hybrid approach.

However, more often, these random sampling-based methods lead to sub-optimal ranking accuracy. There is usually a trade-off between time and accuracy with higher accuracy requiring more samples, thus higher execution time and vice versa. In this chapter, we propose the use of Graph Neural Networks (GNN) to solve the problem of approximation of shortest-path based centrality measures with higher ranking accuracy in a low amount of time. GNN is a type of neural network architecture that leverages graph structure and node/edge feature information to learn node or graph

representations that can be used for many different downstream tasks. The general principle behind GNNs is the node feature aggregation scheme along the edges in the graph. In a multilayer GNN model, each node aggregates features of its neighbors, which lie along all the paths starting or ending at the given node. With repetitive aggregation, the resulting node representation captures the structural information of its neighborhood. One prominent example is the use of graph representations to classify input graphs based on their structure [Xu et al. 2019; Errica et al. 2020; Loukas 2020; Hu et al. 2019]. Building on capability of GNN to learn the graph structure, we propose a novel selective feature aggregation scheme based on the shortest paths in the graph. The input graph is preprocessed, and the adjacency matrix is modified such that the node aggregate features over multiple hops along possible shortest paths in the graph. We then use a ranking based loss to learn a scoring function that maps the aggregated information of the node to a score correlated with the centrality measure scores.

We propose two variants of the model GNN-Bet and GNN-Close for approximating betweenness and closeness centrality, respectively. As per our knowledge, we are the first to propose robust GNN based algorithms for this task. One of the significant benefits of our model is that we can use the computational power of GPUs and once trained, the inference time of our model is very small, in order of milliseconds. Another benefit of our approach is that the model training is inductive, which means that it can be trained on one set of graphs and can be evaluated on another set of graphs with varying structures. The trained model can be used to perform ranking approximations for both static graphs as well as dynamic graphs (with multiple snapshot representation). We demonstrate that our approach dramatically outperforms current techniques (while taking less amount of time) through extensive experiments on a series of synthetic and real-world datasets. For betweenness approximation on static graphs, our model is up to 37 times faster than other methods while providing higher ranking performance. We also show that our proposed betweenness variant of the model is suitable for betweenness approximations for dynamic graphs and is up to 14 times faster in providing node ranking approximations. Closeness approximation variant of our model takes a similar time to the comparison method while providing comparable or better approximations.

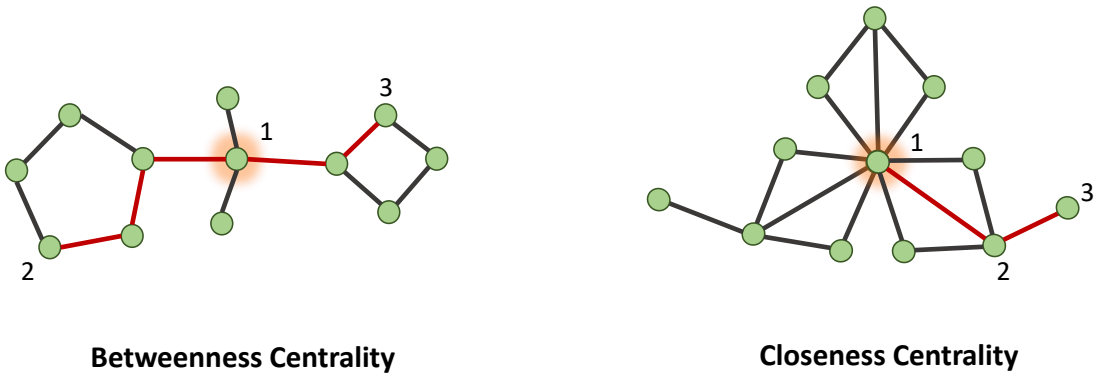


Figure 4.1: Illustration of betweenness and closeness centrality in a graph.

4.2 Preliminaries

4.2.1 Betweenness Centrality

One of the ways a node can be ranked is based on its ability to control the spread of information between other nodes. Ranking of nodes based on this criterion is called betweenness centrality[Freeman 1977; Brandes 2001; Newman 2005]. The main idea behind betweenness centrality is that a given node is central if many shortest paths pass through it. A high value of betweenness centrality for a node means this node lies on many shortest paths between other nodes and helps to pass information between other nodes. In other words, these nodes act as "bridges" in the graph. Removing these nodes from the graph will lead to longer paths for information to travel between other nodes. For a given graph $G = (V, E)$, betweenness centrality of a node v is defined as

$$BC(v) = \sum_{u \neq v \neq w} \frac{\sigma_{uw}(v)}{\sigma_{uw}} \tag{4.1}$$

where σ_{uw} denotes the total number of shortest paths from node u to w and $\sigma_{uw}(v)$ denotes the number of paths from u to w going through node v.

4.2.2 Closeness Centrality

Closeness centrality [Beauchamp 1965; Freeman 1978; Bavelas 1948] gives a measure of how close a given node is to other nodes. Higher closeness centrality means that

the node can spread information easier to other nodes. The closeness centrality of a node v can be defined as

$$CC(v) = \frac{n-1}{\sum_{u \in V} d(v,u)} \tag{4.2}$$

where $d(v,u)$ denotes shortest distance from node v to u.

In simple terms, closeness centrality is proportional to the average of shortest distances from node v to other nodes.
Both betweenness and closeness centrality require calculation of the shortest paths in the graph. Calculating the exact centrality value requires time complexity of $\Theta(|V||E|)$, and thus it is prohibitive for large graphs. To overcome this issue, researchers have proposed approximations algorithms based on sampling techniques. However, these are either less efficient or suboptimal or both. For more details, please refer to Section 4.5.

4.2.3 Graph Neural Networks

In recent years, many neural network models[Duvenaud et al. 2015][Kipf and Welling 2017; Chen, Ma, and Xiao 2018; Velickovic et al. 2018b] have been proposed for graph-structured data and generally known as Graph Neural Networks (GNNs). In all of these models, the structure of the graph is used to aggregate the feature information of nodes and edges[Hamilton, Ying, and Leskovec 2017]. Based on the feature aggregation pattern, a neural network is trained to predict node labels, edge existence probabilities between two nodes and so on. All of these existing models can be generalized under a single common Message Passing Neural Network (MPNN) framework [Gilmer et al. 2017].

Framework Design

Let $G = (V, E)$ denote the graph with given node feature vectors as X_v for $v \in V$. A GNN is designed in such a way that it takes the graph G and feature information matrix of nodes/edges as input and is trained against some arbitrary loss function designated for a specific task. Similar to any other neural network architecture, GNN has a predetermined number of layers decided based on the multitude of factors like task in hand, characteristics of input graphs, the diameter of graph etc.

At any given GNN layer, each node aggregates the features of its neighbors, which can also be considered as a message passing phase. Node feature vector is updated by combining its own feature vector with aggregated features from its neighbors. This aggregation operation is repeated for each layer in GNN. Mathematically, the aggregation operation can be formulated as follows.

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}(\{h_u^{(k-1)} : u \in N_v\}) \tag{4.3}$$

$$h_v^{(k)} = \text{COMBINE}^{(k)}(h_v^{(k-1)}, a_v^{(k-1)}) \tag{4.4}$$

In equation (4.3), the feature vectors h_u of neighboring nodes are aggregated for k-th layer. The AGGREGATE operation can vary based on the model requirement and can represent summing, averaging, or max-pooling of feature vectors. In equation (4.4), the aggregate feature vector is then added to the node feature vector. At the end of each layer, the node has cumulative feature information of the node's neighbors in the current layer and all previous layers. Then, the aggregated features are mapped to a learnable weight matrix and a non-linear transformation like $ReLU$ is used. The output of each layer is then forwarded as the input to the next layer. After K number of iterations, the node embedding vector at the final layer captures the structural as well as node feature information of all neighbors from 1-hop distance to K-th hop distance.

Our proposed GNN framework is constructed following a similar design as mentioned above. In [Wu et al. 2019a], the authors propose a simplified GNN model by removing weights and non-linearity from layers. The simplified model has similar performance to the standard GNN model in classification tasks. However, in our proposed framework, we keep the learnable weights and non-linearity between neural network layers, as we found that the model performed better in this case.

Message Passing in GNN

As discussed earlier, each node aggregates feature information of its neighbors for multiple hop distances. This aggregation scheme resembles breadth first search (BFS) from a node and the feature information flow through all available paths in the graph. Both betweenness centrality and closeness centrality measures, in their calculation, aggregate node contributions by performing BFS (to find shortest paths) from each node. Similarly, in our proposed message passing scheme, we restrict the flow of feature information along certain paths, which are more likely to be the shortest paths in the graph.

Most straightforward formulation to aggregate feature information is multiplying the adjacency matrix to the feature matrix. As rows of adjacency matrix denote indices of node's neighbors, the process of matrix multiplication is similar to the embedding lookup of neighbors from the feature matrix and adding them together. To implement our message passing scheme, we modify the adjacency matrix and perform feature aggregation operation. In-depth details of the construction of our model with the proposed scheme are explored in Section 4.3.

4.3 Proposed Framework

In section 4.2, we observe the similarity in betweenness/closeness centrality formulation and aggregation scheme of GNNs. Taking advantage of this similarity, we propose two variants of GNN-based models that can be trained to approximate betweenness centrality (GNN-Bet) and closeness centrality (GNN-Close) in the graphs. In message passing scheme of GNNs, we observe that each node accumulates the feature vectors of its multi-hop neighbors with the increase in the number of layers or conversely, each node's feature is spread across the graph with each consecutive layer. We use this feature information flow to model paths in the graph. A node that is connected to many other nodes through paths of varying length is able to accumulate more features across the graph.

As both betweenness centrality and closeness centrality measures are based on paths in the graph, we can use GNNs to model the approximation of their calculation. Some other ranking methods like Eigenvector centrality, PageRank centrality are based on iterative methods on the graph. In this chapter, we have not explored the approximations of these types of centrality measures.

The general construction of our proposed framework is similar to other GNNs; however, there are two key differences in our framework, which aid to approximate betweenness and closeness centrality:

(i) First major difference is the use of constrained message passing scheme. In calculations of betweenness and closeness centrality, we do not consider all paths between nodes but shortest paths. In order to approximate these centrality measures in our model, we restrict the feature information flow through edges that lie along the shortest paths. Unlike other GNN frameworks, where regular adjacency matrix is used, we use a modified adjacency matrix such that the feature aggregation in each layer is confined along possible shortest paths. The modification procedure for betweenness variant and closeness variant differs slightly based on the feature gathering scheme.

(ii) The second main difference is that we do not add the node's own feature vectors with the current layer neighborhood feature aggregation vector. This avoids cumulative feature aggregation with subsequent layers and at each layer, the node has unique information about its neighborhood at k-th hop.

4.3.1 Zero Shortest Path Nodes

Both betweenness centrality and closeness centrality measures are based on the calculation of the shortest paths. In our proposed model, we identify beforehand the nodes which have no shortest paths going through them. There are two possible ways to identify such nodes:

(1) When a node has no path going through it. This condition arises when the node either does not have incoming neighbors or does not have outgoing neighbors. For example in Figure4.2, node 1 & 5 have no incoming neighbors and node 12 & 13 have no outgoing neighbors. Hence these nodes have no paths going through them.

(2) When a node's neighbors form cliques such that there is no shortest path going through the node. To identify such nodes, we look at incoming neighbors and outgoing neighbors of a given node. Then we check of there are edges from each incoming neighbor to all outgoing neighbors. For nodes where all incoming neighbors are directly connected to outgoing neighbors, there will be no shortest paths going through them. For instance in Figure4.2, node 2 has incoming neighbor node 1 and outgoing neighbor node 3. However, node 1 is directly connected to node 3, therefore there is no shortest path going through node 2.

With following above mentioned criteria, we can identify nodes with no shortest paths going through them. We refer to these nodes collectively as N_{zp}. In general GNN aggregation scheme, a node aggregates the feature of neighboring node N_k at k-th hop via a path going through sequence of neighboring nodes lying on consecutively decreasing hop distance from source node i.e $(N_k \rightarrow N_{k-1} \rightarrow N_{k-2} \rightarrow \ldots \rightarrow N_2 \rightarrow N_1)$. In order to restrict feature information flow along possible shortest paths, we can modify the aggregation scheme such that nodes N_{zp} do not act as intermediary nodes and no feature information of can flow through the paths that are passing through the nodes N_{zp}.
In the rest of the section, we provide details about the construction of both betweenness and closeness variant.

4.3.2 Betweenness Variant

In a given graph, a node with high betweenness centrality has a large number of shortest paths going through it, and these paths can be considered as a combination

of a set of incoming paths and a set of outgoing paths to the node. In betweenness variant GNN-Bet, we use a selective feature aggregation scheme to estimate the number of shortest paths through the node. Incoming paths and outgoing paths are considered separately, and feature aggregation for both types of paths are performed in parallel. The adjacency matrix of the graph is used for outgoing paths and the transpose of adjacency matrix for incoming paths.

Adjacency Matrix

The first step is to identify the nodes N_{zp}, which have no shortest paths passing through them (as discussed previously in Sec 4.3.1). During the feature aggregation step, feature information should not flow through these nodes. To achieve that, we set the rows corresponding to nodes N_{zp} in the adjacency matrix and its transpose to zero (refer Figure.4.2). With this modification, nodes N_{zp} have no entries for neighbors in their corresponding rows in the adjacency matrix. Hence during the aggregation step, these nodes do not receive any feature information from other nodes and have zero feature information at the output of the layer. This helps to avoid feature flow information through paths that are passing through such nodes as intermediary nodes. Please note in during model initialization all nodes are assigned unique embeddings. In the aggregation step of the first layer, N_{zp} nodes pass their own feature information to neighboring nodes(path length is one). However, in subsequent steps, N_{zp} nodes act as an intermediary(for path length 2 or more) and do not propagate any feature information.

We denote the new modified adjacency matrix as $A_{mod-row}$. The nodes N_{zp} have zero betweenness as per the definition, so with zero feature information, during training, the model learns to distinguish their ranking with respect to other nodes with aggregated feature information.

Model Layer

Initial features of nodes are embedding lookup from the weight matrix of the first layer. At the k-th layer, node aggregates features of its k-hop neighbors. In our model, we define aggregation as the sum of feature vectors. At each layer when the feature matrix is multiplied with the adjacency matrix, each node sums up the features of all of its neighbors. Then the aggregated node features from each layer are mapped to a Multilayer Perceptron (MLP) unit to output a vector whose values contain a single score corresponding to the node. This MLP learns to predict a single score based on the input features. Single MLP unit is used to output scores for all layers. The output score of all layers are summed separately for incoming

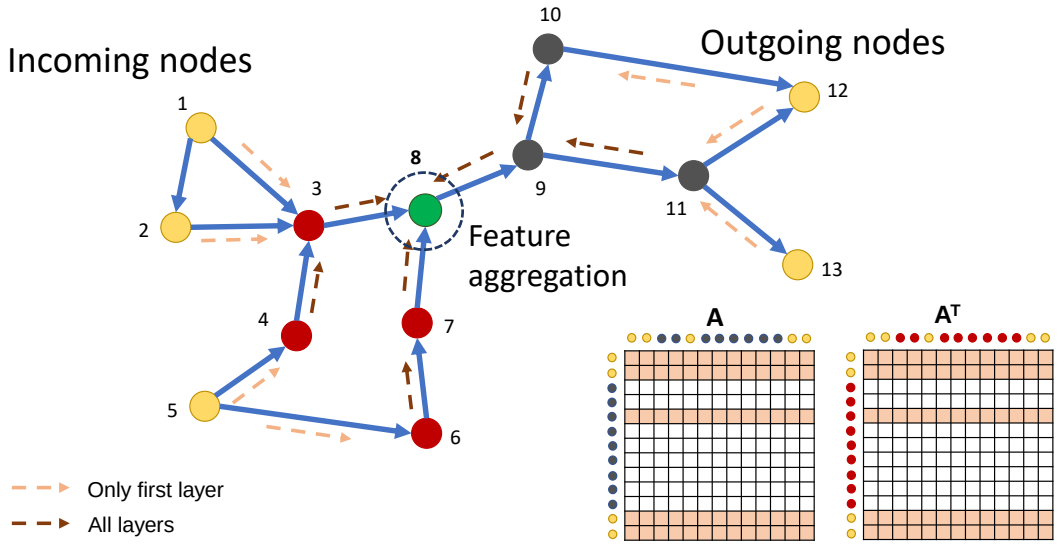


Figure 4.2: Figure shows aggregation scheme for betweenness variant on a directed graph. Features of nodes on both incoming and outgoing paths are aggregated at green node. Dashed arrows show direction of feature flow along the edges. Nodes lying on incoming paths are marked red and outgoing paths marked as black. Yellow nodes do not have shortest paths going through and the corresponding rows are set as zero.

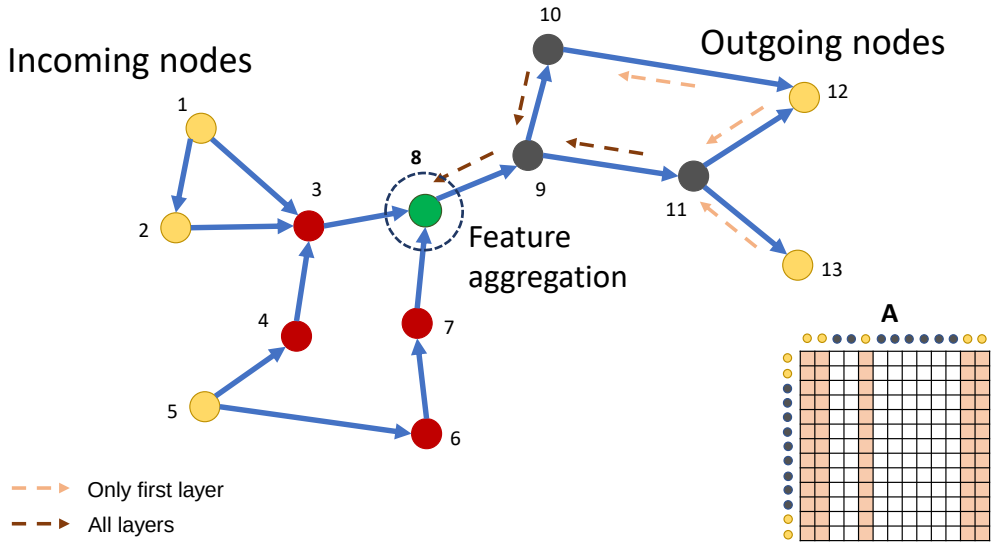


Figure 4.3: Figure shows aggregation scheme of closeness variant along outgoing paths on a directed graph. Features are being aggregated at green node. Dashed arrows show direction of feature flow along the edges. Yellow node do not have shortest paths going through them and the corresponding columns are set as zero.

and outgoing paths for a node. In our experiments, we use an MLP unit comprising three fully connected layers with ReLU as non-linearity.

The feature vectors of in-degree neighbors and out-degree neighbors are accumulated for multiple hops using two matrices. This aggregation approximates the information of incoming paths and outgoing paths to a given node. Hence using our proposed framework, we get two scores for all in-degree neighbors and out-degree neighbors. As the number of paths passing through a given node is the combinations of incoming paths and outgoing paths through the node, the in-degree and out-degree scores are multiplied in order to get the final score for each node. Figure 4.4 shows the schematic diagram of the betweenness variant of the model. Pseudo-code of the forward propagation of the model with feature aggregation scheme is presented in Algorithm 1. At line 1 and 2, adjacency matrix and its transpose is modified (row-wise) to get input matrices $\tilde{A}_{\text{out-degree}}$ and $\tilde{A}_{\text{in-degree}}$ to GNN layer. For each k-th layer {k = 1...K}, $W^{(k)}$ represents the weight matrix and $H^{(k-1)}$ represents the output from the previous layer. MLP is used to map the output node embeddings from each layer to a score vector $S^{(k)}$. Scores are combined from each layer as represented in the line 9-11 to get final centrality scores for all the nodes.

Algorithm 1: GNN-Bet (Forward propagation)

Input : Directed Graph adjacency matrix A; depth K; non-linearity ReLU; weight matrices $W^{(k)}$

Output: Betweenness Centrality value vector, $S_{(Bet)}$

1 $\tilde{A}_{\text{out-degree}} \leftarrow \text{ModifyAdjacencyRow}(A)$
2 $\tilde{A}_{\text{in-degree}} \leftarrow \text{ModifyAdjacencyRow}(A^T)$
3 **for** $k = 1...K$ **do**
4 $\quad H^{(k)}_{\text{out-degree}} \leftarrow \text{ReLU}\ (\tilde{A}_{\text{out-degree}} H^{(k-1)}_{\text{out-degree}} W^{(k)})$
5 $\quad H^{(k)}_{\text{in-degree}} \leftarrow \text{ReLU}\ (\tilde{A}_{\text{in-degree}} H^{(k-1)}_{\text{in-degree}} W^{(k)})$
6 $\quad S^{(k)}_{\text{out-degree}} \leftarrow \text{MLP}(H^{(k)}_{\text{out-degree}})$
7 $\quad S^{(k)}_{\text{in-degree}} \leftarrow \text{MLP}(H^{(k)}_{\text{in-degree}})$
8 **end**
9 $S_{\text{out-degree}} \leftarrow \sum_{k=1..K} |S^{(k)}_{\text{out-degree}}|$
10 $S_{\text{in-degree}} \leftarrow \sum_{k=1..K} |S^{(k)}_{\text{in-degree}}|$
11 $S_{(Bet)} \leftarrow S_{\text{out-degree}} \times S_{\text{in-degree}}$

Algorithm 2: GNN-Close (Forward propagation)

Input : Directed Graph adjacency matrix A; depth K; weight matrices $W^{(k)}$

Output: Closeness Centrality value vector, $S_{(Close)}$

$A_{\text{mod-col}} \leftarrow \text{ModifyAdjacencyCol}(A)$
$H^{(1)} \leftarrow AW^{(1)}$
$S^{(1)} \leftarrow \text{MLP}(H^{(1)})$
for $k = 2...K$ **do**
  $H^{(k)} \leftarrow \text{ReLU}(A_{mod-col}H^{(k-1)}W^{(k)})$
  $S^{(k)} \leftarrow \text{MLP}(H^{(k)})$
end
$S_{(Close)} \leftarrow \sum_{k=1..K} |S^{(k)}|$

4.3.3 Closeness Variant

Closeness centrality has two variations: incoming closeness centrality and outgoing closeness centrality, based on consideration of incoming paths or outgoing paths for the nodes. We focus on the outgoing closeness centrality and use a simple adjacency matrix as input in our model. In case incoming closeness centrality is to be approximated, transpose of adjacency matrix can be used in the model.

For closeness variant of model GNN-Close, we consider paths that originate from source node and end at other nodes. The paths in the graph, which go through N_{zp} are not considered, so we need to identify N_{zp} in this case as well. Unlike betweenness centrality, nodes N_{zp} can have non-zero closeness centrality values, so we use different adjacency matrix modification scheme.

Adjacency Matrix

Similar to the betweenness variant, we identify the nodes which have no shortest paths going through them. To ensure no features flow through edges of these nodes to other nodes, we modify the adjacency matrix by setting the columns corresponding to these nodes to zero (refer Figure. 4.3). We refer to this matrix as $A_{mod-col}$. This operation removes the entries of N_{zp} nodes as neighbor of other nodes and avoids any feature aggregation through them when the path length is 2 or more. However, for the first layer, we still aggregate N_{zp} nodes own features. Therefore, we use the normal adjacency matrix for the first layer and modified adjacency matrix for subsequent GNN layers. To summarize, with this scheme, N_{zp} pass their own feature

information to other neighbors in first iteration but do not act as intermediary for further feature aggregation in later iterations.

Model Layer

As we use an unchanged adjacency matrix in the first layer, nodes aggregate features of all its neighbors. For further layers, we use a modified adjacency matrix and restrict the feature aggregation. Aggregated features of each layer are mapped to an MLP unit to output scores for all layers for each node. Then for each node, output scores from each layer are summed together to get a final score. The output of the model is a score vector. The schematic diagram of GNN-Close is shown in Figure 4.5 and pseudo-code for the model is described in Algorithm 2. At line 1, input adjacency matrix is modified (column-wise) to output $A_{mod-col}$. Line 2 and 3 represent feature aggregation in the first layer with the normal adjacency matrix, the output of which is mapped to the MLP layer. For the rest of the layers, the modified adjacency matrix is used for feature aggregation. Layerwise score vectors are then combined to get final closeness centrality score vector $S_{(Close)}$ at line 8.

4.3.4 Loss Function

We use a ranking loss function to calculate the loss for scores predicted by model with respect to actual betweenness or closeness centrality scores. Ranking loss functions are commonly used in recommender systems [Chen et al. 2009][Burges 2010]. Using this function helps the MLP to learn to predict the score based on the neighborhood of the node at each layer. For the model score, $S_i^{(m)}$ and actual betweenness score $S_i^{(b)}$, the loss function is defined as follows,

$$\text{Loss}\,(x, y) = \max\,(0, -y * (S_i^{(m)} - S_i^{(b)}) + \text{Margin}), \tag{4.5}$$

$$y = \begin{cases} 1 & \text{if } S_i^{(m)} \text{ should be ranked higher than } S_i^{(b)}. \\ -1 & \text{if } S_i^{(b)} \text{ should be ranked higher than } S_i^{(m)}. \end{cases}$$

4.4 Experiments

In this section, we present the details of the training and testing of the model. We evaluate our model on synthetic graphs as well as real-world graphs and compare its performance with other betweenness and closeness approximation techniques.

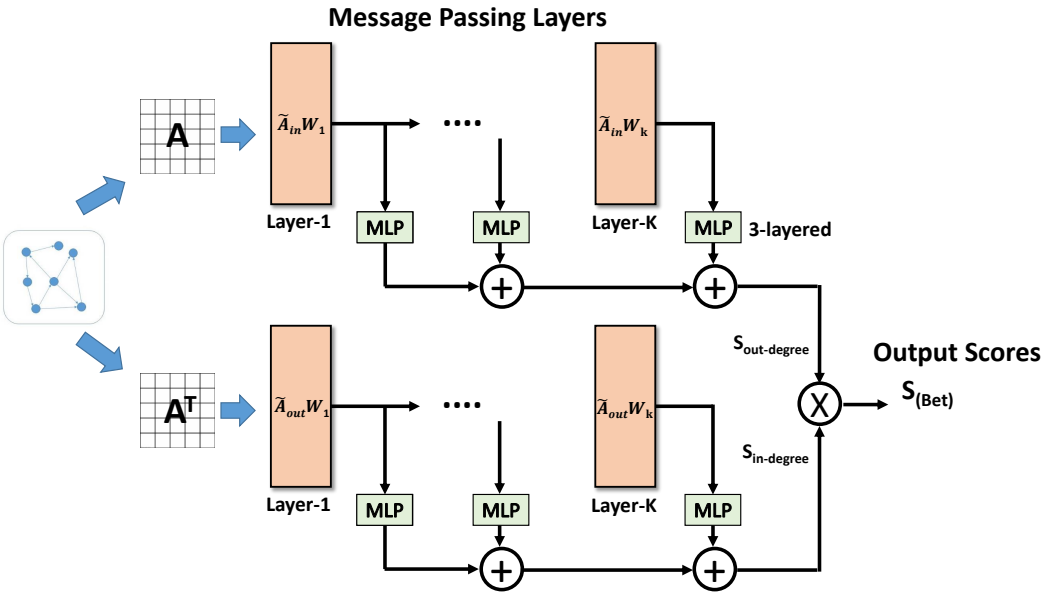


Figure 4.4: Figure above shows the model of betweenness variant. Graph input is represented as an adjacency matrix. There are two parallel aggregation pipelines for incoming and outgoing paths for nodes using adjacency matrix and its transpose respectively. At each layer, node features are aggregated and the output is mapped to a MLP layer. The output of MLP are added together and then subsequently multiplied to get final output scores of nodes.

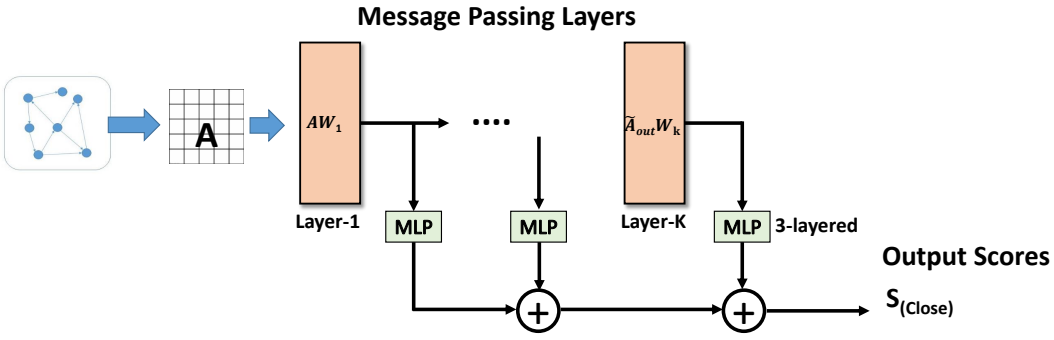


Figure 4.5: Figure shows the model of closeness variant. Graph input is represented as an adjacency matrix. For first layer simple adjacency matrix is used and for further layers modified representation of adjacency matrix (for closeness variant) is used. Feature aggregation is performed at each layer and layer output is mapped to a MLP layer. Sum of output vectors from MLP give final node ranking vector.

4.4.1 Training Setup

Hardware and Software Setup

All experiments were conducted on Intel Core i7-8700K CPU @ 3.70GHz machine with 12-cores, 64 GB RAM and a single NVIDIA GeForce GTX 1080Ti GPU with 12 GB graphics memory. Software frameworks used are `PyTorch`[Paszke et al. 2017] for implementing the proposed model, `NetworkX` [Hagberg, Schult, and Swart 2008] and `NetworKit`[Staudt, Sazonovs, and Meyerhenke 2016] for graph generation and betweenness calculations.

Hyper-parameters

The model size is determined by the largest training/test graph size. For smaller graph inputs, the adjacency matrix of the input graph is placed along diagonal and the rest is padded with zeros. We train the model with Adam as an optimizer with a learning rate parameter set to 0.005. The number of GNN layers in our model is set to 4. Calculation of loss value during training requires comparison of node pair rankings, but the total number of combinations of all node pairs can be really large ($\frac{n(n-1)}{2}$ with n nodes) and computationally prohibitive. In order to avoid this, we randomly sampled node pairs equal to 20 times the number of nodes.

For the betweenness variant, we have used dimension embedding of 12 (limited by GPU memory). For closeness centrality, we set dimension of embedding as 15.

Training

Given set of graphs are divided into training dataset and test dataset. For graphs in both datasets, exact betweenness centrality and closeness centrality values of nodes are calculated. These values are used in target vectors to train the model and to evaluate the performance of the model during inference time for the test dataset. Self-loops are removed from the input graphs while multi-edges are kept, if present in the graphs. We train the model for 5-10 epochs and training time is approximately 10-15 minutes. While training the model, it is possible to have different sizes of graphs for training and test datasets. To accommodate all the graphs, the model size is set to be equal to or greater than the number of nodes of the largest graph in training and test datasets combined. The adjacency matrices of smaller input graphs placed along the diagonals and rest dimensions are padded with zeros to make its size compatible to the input layer.

Note that the test graphs are totally unseen during the training stage (inductive settings). As the model is trained, it learns to map aggregated features for each node via MLP to ranking scores. Once trained, the model is able to take new graphs with different sizes or structures as input and provide node ranking approximations. Hence the model is very robust and can be deployed in various scenarios.

Evaluation Measure

In order to evaluate the ranking quality of model output, we use Kendall's Tau rank correlation coefficient (refer as KT Score)[Kendall 1938] as a ranking measure. For any given pair (x_i, y_i) and (x_j, y_j), where $i < j$, are said to be concordant if the ranks of both elements in both pairs agree, i.e. both $x_i < x_j$ and $y_i < y_j$ or both $x_i > x_j$ and $y_i > y_j$. If the ranks don't agree, then they are said to be discordant. For two given lists with n items each, let's N_c be the total number of concordant pairs and N_d be total number of discordant pairs, then Kendall's ranking coefficient is calculated by:

$$\tau = \frac{N_c - N_d}{\frac{n(n-1)}{2}} \tag{4.6}$$

τ denotes Kendall's Tau ranking coefficient and its value varies in the range $-1 \leq \tau \leq 1$. Value of 1 means all pairs are concordant (same order) and -1 means all pairs are discordant (opposite order). In our case, we compare the ranking of model output for nodes to the ranking of their real centrality values.

4.4.2 Dataset Preparation

To test the efficacy of our model on various graphs with unique structural properties and show that the model is able to generalize well, we use extensive set of synthetic as well as real-world graphs. We evaluate and compare our proposed model to other betweenness and closeness approximation methods on these graphs.

Synthetic Datasets

For synthetic datasets, we generate three classes of directed graphs namely, Erdos-Renyi random graphs, Scale-free graphs, and Gaussian random partition graphs. Sizes of graphs vary from 50,000 to 100,000 nodes. For each class, 15 graphs are generated with the number of nodes and generation parameters set randomly to incorporate structural variations of the dataset. More details of graph generation parameters and statistics of graphs are provided in Appendix A.1. For training of

Table 4.1: Summary of real world datasets used

Dataset	#Nodes	#Edges
Wiki-Vote	7,115	103,689
P2p-Gnutella31	62,586	147,892
Soc-Epinions	75,879	508,837
Soc-Slashdot	77,350	516,575
Ego-Gplus	107,614	13,673,453
Email-EuAll	265,214	420,045
Web-Google	875,713	5,105,039
Wiki-Talk	1,140,149	7,833,140

the model, 5 graphs are chosen to create 500 training samples by randomly permuting the node sequence in adjacency matrices and their corresponding target centrality vector. We train the model on the training set and evaluate the performance on 10 graphs in the test set.

Real-world Datasets

For model evaluation on real-world datasets, we have taken 8 datasets from Stanford Large Network Dataset (SNAP) website[1]. Table 4.1 shows the details for these datasets. Real-world graphs have varying structural properties. Under the weak assumption that real-world graphs have some scale-free graph properties, the model is trained on synthetic scale-free graphs. To create training dataset, 5 synthetic directed scale-free graphs of 100,000 nodes each are generated using `NetworkX`. From these 5 graphs, 250 adjacency matrices are obtained by permuting the node sequences and corresponding betweenness or closeness centrality vectors. 200 of these adjacency matrices are used for training and 50 for validation.

4.4.3 Static Betweenness Approximations

We compare the ranking performance and execution time comparison of our model GNN-Bet to the following betweenness approximation methods for static graphs. Parameters used in these methods are as follows:

- Geisberger et al.[Geisberger, Sanders, and Schultes 2008]: referred as **GS**. For **GS**, we need to provide the number of random nodes to be sampled. We set it to 8,192 which is the maximum number of samples used in the literature even for large graphs.

[1]SNAP Dataset: https://snap.stanford.edu/data/index.html

- Riondato et al.[Riondato and Kornaropoulos 2016]: referred as **RK**. Two parameters are required, $\delta = 0.1$ and $\lambda = 0.005$. These values are used by the authors in their paper. However, we found that **RK** was taking too much time for approximation calculations with $\lambda < 0.02$, sometimes approaching/exceeding the exact betweenness centrality calculation time. So far large graphs, we show results for $\lambda = 0.02$ marked with '$*$'.

- Borassi et al.[Borassi and Natale 2019]: referred as **BN**. Parameters are set as $\delta = 0.1$ and $\lambda = 0.005$ as used in the paper.

In Table 4.2, we compare the ranking performance of our model with other betweenness approximation methods using KT Score. Similarly, Table 4.3 shows the comparison of execution time in seconds for betweenness approximations. We also show the time taken by exact betweenness centrality computation algorithm provided by Brandes[Brandes 2001] as a reference to compare all approximate algorithms. Execution time of our model is the sum of time taken to identify nodes with zero betweenness and model inference time. For other algorithms, the execution time is shown. For synthetic graphs, the average KT score (in Table 4.2) and the average time taken (in Table 4.3) for 10 test graphs are calculated and presented.

KT Score Comparison

Table 4.2 shows KT score for betweenness approximation values for our model and other comparison methods. For real-world graphs, all methods have good ranking performance but the KT Score drops for larger graphs. However, **GS** has consistently good performance compared to **RK** and **BN** for all graph sizes. For synthetic graphs, all methods show high ranking performance on ER graphs, while lower KT Scores for SF graphs and GRP graphs. Our model GNN-Bet, consistently performs well on real-world as well as synthetic datasets with up to 44% improvement for real-world graphs and up to 70% for synthetic graphs with respect to the best KT Score on comparison methods.

Execution Time Comparison

Execution time comparison is presented in Table 4.3. We observe that **GS** provides good approximations with reasonable execution time, which is lower than exact betweenness values calculations. For **RK**, the execution frequently approaches/exceeds exact betweenness calculation time if we set parameters for better approximations. The method is therefore good if higher error tolerance is allowed. **BN** is the fastest

Table 4.2: KT ranking scores comparison static betweenness approximation on real world and synthetic datasets. Highest KT Score among comparison methods(GS, RK and BN) is underlined and best ranking performance among all is highlighted with bold font. Performance gain for GNN-Bet is compared with respect to the best KT Score among comparison methods.

Dataset	**KT Score**				
	GS	**RK**	**BN**	**GNN-Bet**	**Gain**
Wiki-Vote	0.866	0.844	0.900	**0.967**	+0.067 (+7.4%)
P2p-Gnutella31	0.849	**0.963**	0.876	0.924	-0.039 (-4.0%)
Soc-Epinions	0.723	0.826	0.687	**0.891**	+0.065 (+7.8%)
Soc-Slashdot	0.766	0.793	0.639	**0.909**	+0.116 (+14.6%)
Ego-Gplus	0.817	0.673	0.515	**0.821**	+0.004 (+0.4%)
Email-EuAll	0.889	0.508	0.380	**0.990**	+0.101 (+11.3%)
Web-Google	0.755	0.366*	0.498	**0.779**	+0.024 (+3.1%)
Wiki-Talk	0.679	0.204*	0.271	**0.979**	+0.300 (+44.1%)
Synthetic-ER	0.845±0.05	0.829±0.07	0.629±0.14	**0.902±0.03**	+0.057 (+6.7%)
Synthetic-SF	0.464±0.02	0.574±0.03	0.411±0.03	**0.976±0.01**	+0.402 (+70.0%)
Synthetic-GRP	0.562±0.06	0.342±0.17	0.141±0.08	**0.899±0.04**	+0.337 (+59.9%)

among all the comparison methods, taking very little time to output approximation values. However, as evident from Table 4.2, the ranking accuracy drops at the expense of speed. Our proposed model is up to 37 times faster than **BN** for real-world datasets and up to 33 times faster for synthetic graph datasets.

4.4.4 Dynamic Betweenness Approximations

Our model can readily be applied to dynamic graphs, where nodes and edges changes with time. With these changes, we get multiple slices of graphs. Traditionally, we have to re-calculate betweenness centrality on each slice from scratch, which is computationally expensive. More recent methods overcome this limitation by first calculating betweenness for all nodes and then updating the values as the graph changes. However, in practice, this approach is still slower and may not provide good approximations with larger changes in graph. Since our model training is inductive, we can use the trained model for betweenness approximations to get approximations of all slices of dynamic graphs. In this way, our model can provide more accurate and faster approximations.

We compare our model with the algorithm proposed by [Hayashi, Akiba, and Yoshida 2015] referred to as **HA**. In the experiments, for each graph, we initially removed 1000 edges. Then from those 1000 edges, we randomly take 200 edges and add them back to the graph structure while removing 200 different edges from the

Table 4.3: Time comparison for static betweenness approximation methods. Lowest execution time among all comparison methods is underlined and bold fonts denotes the fastest method. Speedup gains for GNN-Bet are calculated with respect to fastest among comparison methods.

Dataset	$\textbf{Time}_{\textbf{train}}$(s) GNN-Bet	Time (s)					
		Exact	GS	RK	BN	GNN-Bet	Speedup
Wiki-Vote	633.5	0.8	7.1	1.6	0.2	**0.1**	≈ 2x
P2p-Gnutella31		57.2	18.06	163.4	5.4	**0.2**	≈ 27x
Soc-Epinions		222.1	86.5	567.9	6.8	**0.5**	≈ 14x
Soc-Slashdot		206.7	71.2	510.3	5.96	**0.9**	≈ 7x
Ego-Gplus		3845.7	979.4	7787.5	11.8	**7.2**	≈ 2x
Email-EuAll		1091.5	99.8	985.1	8.1	**1.1**	≈ 7x
Web-Google		52422	2264	8548*	314.4	**8.5**	≈ 37x
Wiki-Talk		32385	624.2	1115.4*	32.4	**3.6**	≈ 9x
Synthetic-ER	726.8	811.6±560.1	206.3±142.0	1354.2±765.7	20.2±3.3	**0.6±0.3**	≈ 33x
Synthetic-SF	317.0	33.81±4.4	5.2±1.7	114.0±30.7	1.8±0.3	**0.3±0.2**	≈ 6x
Synthetic-GRP	757.8	27.8±12.5	2.9±1.4	98.0±30.7	1.5±0.4	**0.6±0.2**	≈ 2x

graph. We repeat this step five times. So at each step, there is a change of 400 edges. We then compare the resulting KT Score and time taken by our model and the comparison method.

Table 4.4 & 4.5 show KT Score and time performance results. Results show significant gains in KT Score (up to 300% improvement) while still being faster (up to 14 times faster). This shows the versatility of the model to be used for both static and dynamic graphs. We note that if the number of edge changes is smaller (number of edge changes in tens or less), **HA** may have execution time comparable to our model. However, with smaller changes in graphs, changes in centrality values are not significant and may not be very useful in real-world applications.

4.4.5 Static Closeness Approximations

In this section, we evaluate our closeness variant of our model with respect to other closeness centrality approximation methods. For this, we compare to the methods proposed by Okamoto et al.[Okamoto, Chen, and Li 2008] and Cohen et al.[Cohen et al. 2014] and refer to the methods as **OC** and **CD** respectively. KT-Score and execution time for algorithms are used as comparison metrics. For **CD**, the parameter for the number of nodes to be randomly sampled is chosen as 100, as used in the paper[Cohen et al. 2014] and we set parameter $\epsilon = 0.001$. The number of nodes to be sampled for **OC** is also set as 100.

For closeness centrality approximations, Table 4.7 shows the results for real-world and synthetic datasets. Our model outperforms **OC** and **CD** for synthetic datasets with performance improvements up to 85% and 214% respectively. For

Table 4.4: KT ranking scores comparison for dynamic betweenness approximation method on real world dataset. Best KT Score are highlighted in bold.

Dataset		**KT Score**					
		Update-1	**Update-2**	**Update-3**	**Update-4**	**Update-5**	**Gain(avg)**
Wiki-Vote	HA	0.8909	0.8964	0.9013	0.9016	0.9031	+0.0647 (+7.2%)
	GNN-Bet	**0.9633**	**0.9636**	**0.9623**	**0.9622**	**0.9655**	
P2p-Gnutella31	HA	0.7233	0.7242	0.7282	0.7269	0.7256	+0.2023 (+27.8%)
	GNN-Bet	**0.927**	**0.928**	**0.928**	**0.928**	**0.929**	
Soc-Epinions	HA	0.5742	0.5719	0.5746	0.5698	0.5706	+0.3117 (+54.4%)
	GNN-Bet	**0.8832**	**0.8837**	**0.8847**	**0.8834**	**0.8846**	
Soc-Slashdot	HA	0.5358	0.5309	0.5368	0.5353	0.5360	+0.3705 (+69.2%)
	GNN-Bet	**0.9048**	**0.9053**	**0.9053**	**0.9057**	**0.9060**	
Ego-Gplus	HA	0.6073	0.6097	0.6126	0.6144	0.6123	+0.2001 (+32.7%)
	GNN-Bet	**0.8111**	**0.8119**	**0.8118**	**0.811**	**0.8111**	
Email-EuAll	HA	0.4312	0.4292	0.4263	0.4273	0.4316	+0.5581 (+130.0%)
	GNN-Bet	**0.9877**	**0.9808**	**0.9889**	**0.9891**	**0.9899**	
Web-Google	HA	0.3001	0.2997	0.2988	0.2998	0.3019	+0.4874 (+162.4%)
	GNN-Bet	**0.7664**	**0.7668**	**0.7673**	**0.7682**	**0.7684**	
Wiki-Talk	HA	0.2389	0.2366	0.2438	0.2402	0.2420	+0.7369 (+306.6%)
	GNN-Bet	**0.9770**	**0.9774**	**0.9773**	**0.9772**	**0.9772**	

the real-world datasets, approximation performance is higher in five out of eight datasets when compared with **OC** and seven out of eight datasets when compared with **CD**. However, our model's execution time is slightly higher than **CD**, though the absolute differences between values are not significant. Compared to **OC**, our model has significantly faster execution time in most of the comparison datasets. In our model implementation, most of the time is taken by the function to find nodes with no shortest path passing through them. As the code is implemented in `Python`, it is relatively slower. The time taken during GNN model inference is only a few milliseconds. Hence, there is a scope to reduce the approximation time even further. Interestingly, we found that the output of **CD** had a negative KT score for synthetic Gaussian Random Partition graphs.

4.4.6 Scalability Tests

In previous sections, we have demonstrated the ranking performance of GNN-Bet & GNN-Close and their robustness with different types of graphs. In this section, we study the effect on time to find zero shortest path nodes and the execution time of the model in forward propagation with respect to the increase in the size of the input graph.

As discussed in Section 3.1, there are two possible ways to identify N_{zp} nodes. First, if either in-degree or out-degree of the node is zero, there is no path passing through the node. Time complexity of finding such nodes is $O(V)$. We identify

Table 4.5: Time comparison for dynamic betweenness approximation methods. Lowest execution values are highlighted in bold.

Dataset		Time(s)					
		Update-1	**Update-2**	**Update-3**	**Update-4**	**Update-5**	**Speedup**
Wiki-Vote	HA	3.42	2.81	3.67	3.87	3.14	≈ 7x
	GNN-Bet	**0.53**	**0.52**	**0.51**	**0.54**	**0.44**	
P2p-Gnutella31	HA	2.07	2.17	2.10	2.07	2.04	≈ 13x
	GNN-Bet	**0.17**	**0.16**	**0.17**	**0.16**	**0.16**	
Soc-Epinions	HA	10.69	12.74	11.51	13.16	10.77	≈ 14x
	GNN-Bet	**0.97**	**0.80**	**0.83**	**0.80**	**0.80**	
Soc-Slashdot	HA	5.79	3.82	4.56	4.20	4.20	≈ 6x
	GNN-Bet	**0.77**	**0.77**	**0.76**	**0.76**	**0.77**	
Ego-Gplus	HA	99.06	74.25	77.25	72.34	77.47	≈ 12x
	GNN-Bet	**6.94**	**6.82**	**6.91**	**6.86**	**6.25**	
Email-EuAll	HA	7.21	8.25	8.22	3.21	7.59	≈ 7x
	GNN-Bet	**0.89**	**0.90**	**0.89**	**0.90**	**1.10**	
Web-Google	HA	**4.56**	**4.53**	**4.57**	**4.51**	**4.61**	≈ 0.8x
	GNN-Bet	5.45	5.48	5.52	5.52	5.56	
Wiki-Talk	HA	3.48	**3.48**	4.04	3.77	**3.67**	≈ 1x
	GNN-Bet	**3.45**	3.50	**3.61**	**3.66**	3.72	

such nodes in first step. Second, if the node's neighbors form cliques such that there is no shortest path going through the node. We identify such nodes by iterating through the remaining nodes $v \in V_{remain}$ from the first step and check if any of the in-degree neighbors N_{in_v} of the node v is not connected directly to all out-degree neighbors N_{out_v}. If such a node in N_{in_v} is found, it means that there is a path going through node v, and the node is not considered to be in N_{zp}. Then iteration over other in-degree nodes is skipped, and the process is repeated for the next node in V_{remain}. Considering the average number of in-degree or out-degree as $N_{in\_avg} = N_{out\_avg} = N_{avg}$, the time complexity of the operation is $O(V{N_{avg}}^2)$. In case of low density graphs, $N_{avg} << V$ and we do not have to iterate over all N_{in_v}, hence the complexity is almost linear (in number of nodes), approaching $O(VN_{avg})$. However, if the graph is dense and approaching a fully connected graph, then $N_{avg} \simeq V$ and the time complexity of the operation becomes $O(V^3)$. To experimentally verify the time scalability of the method to find zero shortest paths nodes, we generate three sets of Erdos-Renyi random graphs with number of edges to number of nodes ratio set to 2, 4 and 6 respectively. This provides a variation in density of the graphs. We chose ER graphs for this experiment for the convenience of setting the desired number of nodes and edges in the generated graphs easily. The number of nodes are varies from a hundred thousand nodes to a million nodes for each set. Figure 4.6 shows the plot of execution time to find zero shortest path nodes with respect to change in the graph size. We observe that execution time grows proportional to the

Table 4.6: KT ranking scores comparison for closeness approximations on real-world and synthetic datasets. Higher KT Score between comparison methods(OC and CD) is underlined and best ranking performance among all is highlighted with bold font. Performance gain for GNN-Close is compared with respect to the best KT Score of comparison methods.

Dataset	KT Score			
	OC	CD	GNN-Close	Gain
Wiki-Vote	0.916	0.567	**0.937**	+0.021 (+2.2%)
P2p-Gnutella31	0.930	0.926	**0.978**	+0.052 (+5.1%)
Soc-Epinions	**0.945**	0.810	0.917	-0.028 (-3.0%)
Soc-Slashdot	0.928	0.778	**0.955**	+0.027 (+2.9%)
Ego-Gplus	**0.964**	0.909	0.923	+0.041 (-4.4%)
Email-EuAll	0.847	0.494	**0.927**	+0.080 (+9.4%)
Web-Google	**0.920**	0.533	0.699	-0.221 (-31.6%)
Wiki-Talk	0.963	**0.979**	0.978	-0.001 (-0.1%)
Synthetic-ER	0.812 ± 0.04	0.651±0.26	**0.907±0.03**	+0.095 (+11.6%)
Synthetic-SF	0.861 ± 0.04	0.287±0.10	**0.903±0.01**	+0.042 (+4.8%)
Synthetic-GRP	0.527 ± 0.18	-0.403±0.11	**0.979±0.01**	+0.452 (+85.7%)

Table 4.7: Time comparison for closeness approximation methods. Lowest execution time among all comparison methods is underlined and bold fonts denotes the fastest method. Speedup gains for GNN-Close are calculated with respect to fastest among comparison methods.

Dataset	$\text{Time}_{\text{train}}$(s)	Time(s)				
	GNN-Close	Exact	OC	CD	GNN-Close	Speedup
Wiki-Vote	349.0	0.6	0.03	**0.02**	0.1	≈ 0.2x
P2p-Gnutella31		42.9	1.3	**0.1**	0.2	≈ 0.5x
Soc-Epinions		160.7	2.1	**0.3**	0.4	≈ 0.7x
Soc-Slashdot		152.7	2.0	**0.3**	0.5	≈ 0.6x
Ego-Gplus		3458.1	4.5	**2.7**	6.2	≈ 0.4x
Email-EuAll		807.6	3.7	0.8	**0.5**	≈ 1.6x
Web-Google		38176.7	18.6	10.3	**5.3**	≈ 1.9x
Wiki-Talk		28792.0	9.7	**2.6**	3.1	≈ 0.8x
Synthetic-ER	371.4	484.1±452.0	3.0 ± 1.1	**0.53±0.36**	0.8±0.6	≈ 0.6x
Synthetic-SF	235.2	26.4±11.4	0.9 ± 0.2	**0.16±0.02**	0.3±0.4	≈ 0.5x
Synthetic-GRP	383.3	28.8±21.1	1.0 ± 0.2	**0.24±0.1**	0.4±0.56	≈ 0.6x

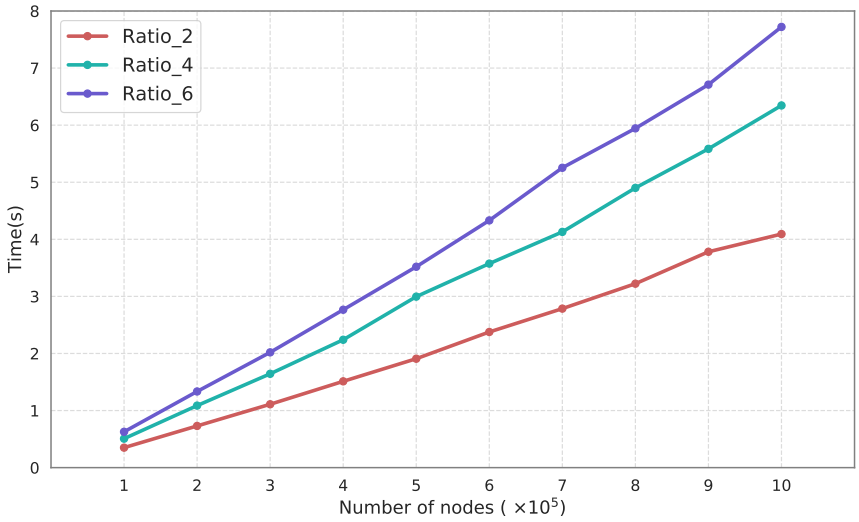


Figure 4.6: Figure depicts the execution time with respect to increase in number of nodes in the input graph. The ratio of number of nodes to number of edges is fixed as 2, 4 and 6 respectively.

number of nodes, and time increments are rather small.

Now we study the scalability of the inference time of model with an increase in size of the input graphs. Here we use the betweenness centrality variant of the model, as calculation of betweenness centrality takes a longer time compared to that of closeness centrality. For static betweenness centrality approximation, **BN** is the fastest betweenness approximation method. In order to compare the speed of approximation of our model(GNN-Bet) with **BN**, we test them on 10 synthetic scale-free graphs with the number of nodes ranging from 100,000 to 1,000,000. We set the size of the model to accommodate for all input graphs (i.e. maximum 1 million nodes). Input adjacency matrices of smaller graphs are padded with zeros to match with model size. Hence the density of all input matrices is proportional to the number of edges in each graph. Therefore, the time taken is approximately proportional to the number of edges and time complexity is $O(|E|)$ (similar to Graph Convolutional Network (GCN)[Wu et al. 2019b]). Figure 4.7 shows the execution time taken by **BN** and betweenness variant of model for all graphs. We observe that in both cases, execution time grows proportional to the size of the graph. Nevertheless, our proposed model GNN-Bet has a lower rate of increment, thus scaling well with the size of graphs and providing faster approximations.

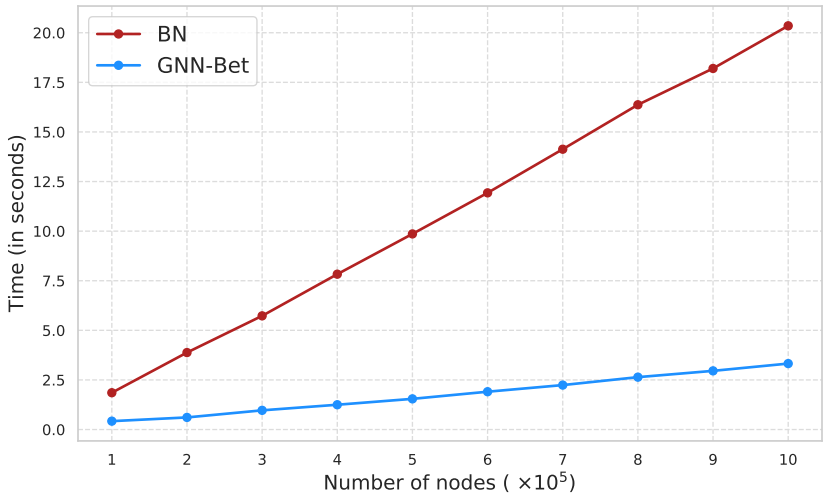


Figure 4.7: Time taken by model as number of nodes increases for synthetic scale-free graphs

4.4.7 Ablation tests

In this section, we conduct a series of experiments to study the effect of hyper-parameters on the model's performance for both variants of our proposed model. As the main parameters of the model are the number of layers and the number of dimensions of the embeddings, we vary both of them and evaluate the performance of the model. We choose to use synthetic datasets for these experiments, as it provides a better understanding of the model's performance on different types of graph structures. The experiments follow the same methodology as discussed earlier.

- *Varying number of layers*: The number of layers in the model affects how much information any given node can accumulate about its neighborhood. With the increase in the number of layers, nodes have access to features of their neighbors located at multi-hop distances away. In this experiment, we vary the number of layers from 1 to 7 for both betweenness and closeness centrality variants. Fig. 4.8 and Fig. 4.10 show Kendall's Tau score for betweenness and closeness variant respectively for all three types of synthetic graphs. We observe that for a lower number of layers model performs poorly, which is within expectation as the nodes aggregation reach is limited. For both variants GNN-Bet and GNN-Close, we observe that an increase in the number of layers leads to better approximations accuracy.

- *Varying embedding dimensions*: While training any neural network, the embedding dimension determines the number of learnable model parameters. An under-parameterized neural network model may not learn very well while the over-parameterized model can overfit the data and have poor generalization ability. In this experiment, we look into the effect of changing embedding dimensions on the performance of our proposed model. We vary the model's embedding dimensions as [2, 4, 8, 12, 16, 20] and evaluate the ranking performance on synthetic graph datasets. Results are plotted in Fig.4.9 and Fig.4.11 for betweenness and closeness variant respectively. With lower dimensions of embedding (2 & 4), we observe lower performance which is natural due to the limited number of parameters for optimal learning.

4.4.8 Ranking performance on top-k nodes

In our main experiments for betweenness and closeness centrality approximations, we have focused on the overall ranking of nodes predicted by the model and compared our results with similar node centrality approximation methods. In addition to this approach, we can also measure the performance of the model on identifying top-k high centrality nodes. We can measure the ranking performance quality using measures like NDCG. In this section, we provide brief experiment results on above mentioned criteria.

Measuring NDCG score for top-k nodes

Normalized Discounted Cumulative Gain (NDCG) is used to measure the quality of search results and is often used in recommender or web search systems [Wang et al. 2013]. The NDCG score value ranges from 0 to 1, with 0 denoting low relevancy to 1 denoting very high relevancy of search results. We use this score to measure the ranking performance of top-k nodes with high centrality values. The value of k is chosen as 100, 1%, 2%, 3%, 5%, and 10% of the total number of nodes in the graph.

Table 4.8, 4.9 & 4.10 show the comparison of NDCG scores for GS, RK, and BN betweenness centrality approximation methods with GNN-Bet respectively. Table 4.11 & 4.12 shows the comparison of closeness centrality approximation method OC and CD with GNN-Close, respectively.

- **Betweenness Centrality Approximations** For synthetic graphs, GNN-Bet performs better or at par with other approximation methods on ER and SF graph datasets. On these two datasets for all methods, the NDCG scores

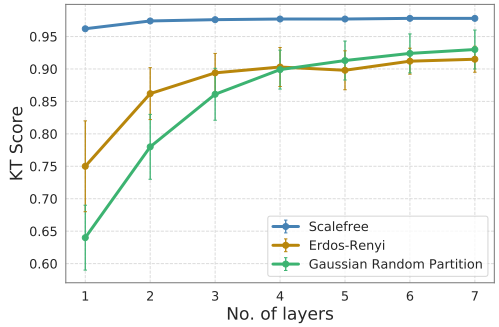


Figure 4.8: Change in KT Score with respect to number of layers for Betweenness Variant

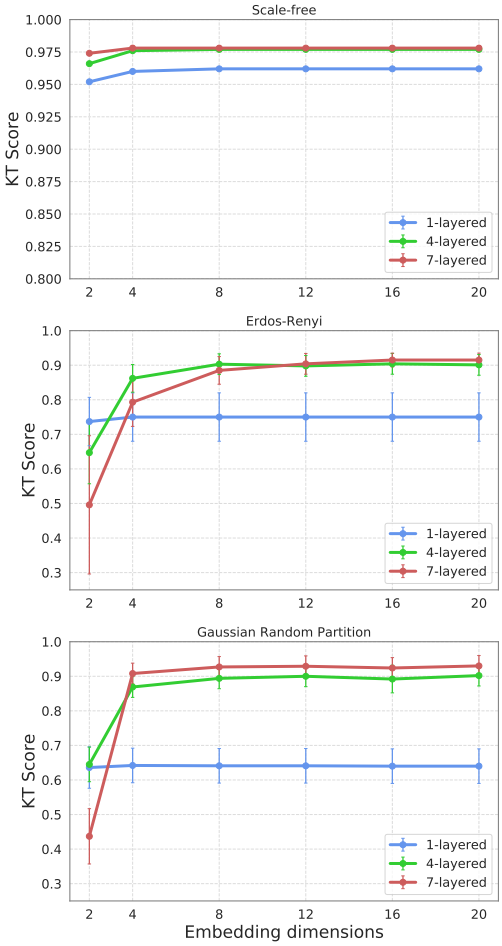


Figure 4.9: Change in KT Score with respect to embedding dimensions for Betweenness Variant

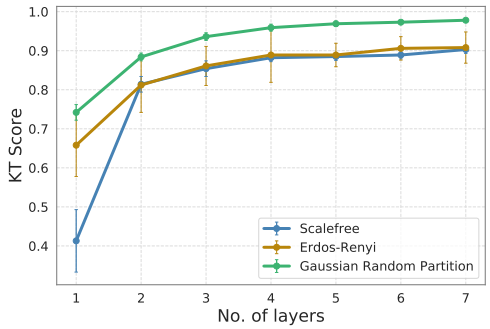


Figure 4.10: Change in KT Score with respect to number of layers for Closeness Variant

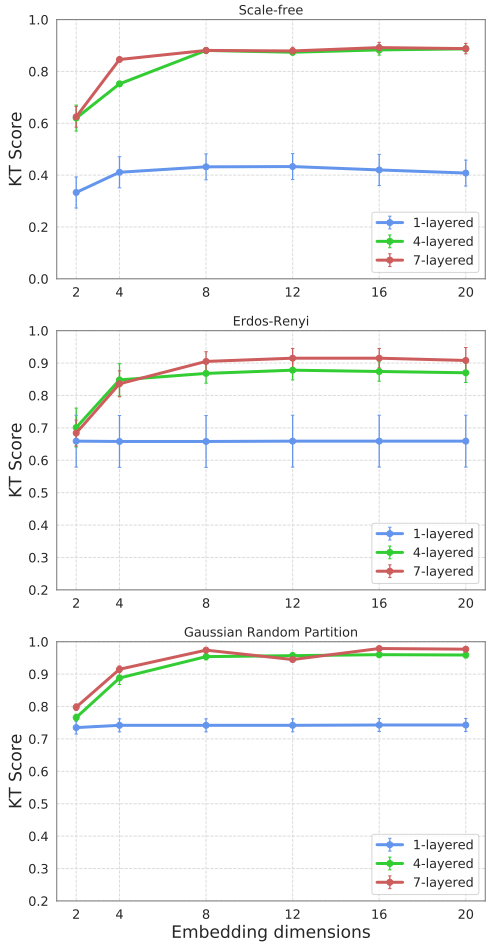


Figure 4.11: Change in KT Score with respect to embedding dimensions for Closeness Variant

remains higher than 0.9. On the GRP graph dataset, the model performs significantly better than other methods, with NDCG scores closer to 1. Among comparison methods, GS performs best with NDCG score higher than 0.8, while for RK and BN, the NDCG scores remains close to 0.6 and 0.3, respectively.

On the other hand, for real-world graph datasets, the NDCG scores of comparison methods GS, RK, and BN remain high for all datasets. GNN-Bet shows good performance on some datasets like Wiki-Vote, P2p-Gnutella31, and Soc-Slashdot while the performance is lower on other datasets. It is to be noted that for inference on real-world datasets, we trained the model on synthetic scale-free datasets.

While analyzing the results, we find that the NDCG scores of comparison methods go down slightly as we increase the value of the k. While for GNN-Bet model, the score values tend to increase as we increase the value of k. These observations also match with our main experiment results, where our model has a high KTScore for ranking all nodes in the graph.

- **Closeness Centrality Approximations** For synthetic graphs, we see a similar pattern where our model has higher or at par performance with OC method on ER and SF graph datasets and significantly higher performance on the GRP dataset. On real-world datasets, the performance of GNN-Close is at par with OC on most datasets while lower in some datasets. However, the approximation method CD does not perform well on this measure and has very low NDCG values for both synthetic and real-world datasets.

Analyzing the above results, we find that our proposed model performs well on synthetic datasets for both centrality measures. While on real-world datasets, the performance of our model is at par or lower compared to other approximation methods. The top-k performance drop of our model on real-world datasets is higher in the case of betweenness centrality than closeness centrality. There are two reasons that can be attributed to these performance characteristics:

(1) In our model training, the loss function does not include a scheme to provide higher weights to high-ranking nodes and calculates pairwise ranking loss for the nodes as a learning measure.

(2) For training on synthetic graph datasets, we could generate a training set by creating distinct graphs with similar graph structural properties, and our model achieves good testing performance. However, for real-world datasets,

it is challenging to generate a training set. Hence, we train our model on the synthetic scale-free graph dataset and inference on all real-world dataset graphs. As the training dataset is significantly different from the real-world testing dataset and may not share the same graph structural properties, the performance on all test datasets may not be optimal.

Model training with real-world datasets

In previous section, we observed that our proposed model does not perform well for predicting top-k ranking on many real-world datasets. This effect is more pronounced with betweenness centrality. To identify the primary reason for worse performance, we switch the training dataset from a synthetic graph dataset to a real-world graph dataset. For each real-world dataset, we create new graphs derived from the original graph by modifying its structure. We use the graph-randomization procedure proposed by [Gkantsidis, Mihail, and Zegura 2003] and implemented in NetworKit [Staudt, Sazonovs, and Meyerhenke 2016]. In this procedure, two edges are selected uniformly at random and nodes are interchanged. This step can be repeated multiple times and is set as a hyper-parameter.

For our experiment, we selected 10% of the total number of edges for each dataset and generated 10 different randomized graphs. For Wiki-Vote, P2p-Gnutella31, Soc-Slashdot, and Email-EuAll datasets, we generated 50 training samples from each graph by permuting the node sequence and created 500 training samples. For Ego-Gplus, Web-Google, and Wiki-Talk datasets, due to system memory constraints, we generated 10 training samples from each graph and created 100 training samples. As GNN-Bet shows significant drop in top-k performance, we focused our experiment on the betweenness centrality case. We trained the model for each real-world dataset with training samples from the derived graph and measured test dataset performance for top-100, top-1%, top-2%, and top-3% NDCG ranking scores.

Table 4.13 shows the comparison of NDCG scores of GNN-Bet when trained on synthetic graph dataset versus real-world graph dataset. We observe significant improvements in NDCG score values for all datasets, especially for datasets with larger training sample sizes. These results show that if our proposed model is trained on graph datasets similar to the test graphs, our model can perform quite well on the top-k ranking task.

4.4.9 Discussion

The experimental results in Section 4.4.3, 4.4.4 & 4.4.5 show our GNN based approach to node ranking in graphs is highly performant and robust. However, like all

Table 4.8: NDCG Score comparison between GS and GNN-Bet of top-k nodes for betweenness centrality.

Dataset	GS					
	Top-100	**Top 1%**	**Top 2%**	**Top 3%**	**Top 5%**	**Top 10%**
Wiki-Vote	0.995	0.997	0.993	0.994	0.993	0.994
P2p-Gnutella31	0.973	0.980	0.980	0.982	0.985	0.987
Soc-Epinions	0.984	0.985	0.985	0.985	0.985	0.984
Soc-Slashdot	0.978	0.982	0.983	0.982	0.982	0.981
Ego-Gplus	0.993	0.992	0.992	0.990	0.989	0.990
Email-EuAll	0.983	0.992	0.993	0.993	0.993	0.993
Web-Google	0.994	0.989	0.989	0.988	0.988	0.988
Wiki-Talk	0.986	0.987	0.985	0.987	0.989	0.991
Synthetic-ER	0.976±0.0102	0.979±0.0083	0.981±0.0069	0.982±0.0062	0.984±0.0056	0.986±0.0047
Synthetic-SF	0.986±0.0149	0.987±0.0133	0.987±0.0134	0.986±0.0133	0.985±0.0132	0.986±0.0132
Synthetic-GRP	0.883±0.0421	0.888±0.0357	0.888±0.0319	0.889±0.0294	0.890±0.0279	0.893±0.0277

Dataset	GNN-Bet					
	Top-100	**Top 1%**	**Top 2%**	**Top 3%**	**Top 5%**	**Top 10%**
Wiki-Vote	0.945	0.951	0.952	0.959	0.964	0.976
P2p-Gnutella31	0.901	0.934	0.947	0.953	0.959	0.967
Soc-Epinions	0.660	0.716	0.751	0.782	0.823	0.873
Soc-Slashdot	0.764	0.862	0.900	0.919	0.934	0.950
Ego-Gplus	0.408	0.488	0.563	0.606	0.649	0.700
Email-EuAll	0.694	0.831	0.841	0.852	0.867	0.868
Web-Google	0.611	0.609	0.652	0.684	0.726	0.788
Wiki-Talk	0.605	0.755	0.765	0.770	0.782	0.807
Synthetic-ER	0.981±0.0192	0.984±0.0166	0.986±0.0151	0.987±0.0140	0.988±0.0120	0.990±0.0095
Synthetic-SF	0.935±0.0770	0.945±0.0555	0.951±0.0452	0.954±0.0409	0.959±0.0342	0.969±0.0266
Synthetic-GRP	0.981±0.0075	0.986±0.0048	0.988±0.0042	0.989±0.0040	0.992±0.0033	0.993±0.0028

Table 4.9: NDCG Score comparison between RK and GNN-Bet of top-k nodes for betweenness centrality.

Dataset	RK					
	Top-100	**Top 1%**	**Top 2%**	**Top 3%**	**Top 5%**	**Top 10%**
Wiki-Vote	0.999	0.999	0.999	0.999	0.999	0.999
P2p-Gnutella31	0.998	0.997	0.996	0.995	0.995	0.995
Soc-Epinions	0.999	0.999	0.988	0.998	0.998	0.997
Soc-Slashdot	0.999	0.999	0.998	0.998	0.997	0.996
Ego-Gplus	0.999	0.999	0.999	0.998	0.997	0.996
Email-EuAll	0.999	0.998	0.997	0.997	0.997	0.997
Web-Google	0.998	0.971	0.956	0.950	0.945	0.918
Wiki-Talk	0.961	0.861	0.852	0.852	0.854	0.859
Synthetic-ER	0.969±0.0269	0.972±0.0249	0.973±0.0230	0.974±0.0221	0.975±0.0215	0.977±0.0199
Synthetic-SF	0.999±0.0003	0.998±0.0004	0.997±0.0007	0.997±0.0006	0.996±0.0009	0.997±0.0006
Synthetic-GRP	0.607±0.2459	0.613±0.2220	0.605±0.2265	0.607±0.2241	0.619±0.2150	0.632±0.2046

Dataset	GNN-Bet					
	Top-100	**Top 1%**	**Top 2%**	**Top 3%**	**Top 5%**	**Top 10%**
Wiki-Vote	0.945	0.951	0.952	0.959	0.964	0.976
P2p-Gnutella31	0.901	0.934	0.947	0.953	0.959	0.967
Soc-Epinions	0.660	0.716	0.751	0.782	0.823	0.873
Soc-Slashdot	0.764	0.862	0.900	0.919	0.934	0.950
Ego-Gplus	0.408	0.488	0.563	0.606	0.649	0.700
Email-EuAll	0.694	0.831	0.841	0.852	0.867	0.868
Web-Google	0.611	0.609	0.652	0.684	0.726	0.788
Wiki-Talk	0.605	0.755	0.765	0.770	0.782	0.807
Synthetic-ER	0.981±0.0192	0.984±0.0166	0.986±0.0151	0.987±0.0140	0.988±0.0120	0.990±0.0095
Synthetic-SF	0.935±0.0770	0.945±0.0555	0.951±0.0452	0.954±0.0409	0.959±0.0342	0.969±0.0266
Synthetic-GRP	0.981±0.0075	0.986±0.0048	0.988±0.0042	0.989±0.0040	0.992±0.0033	0.993±0.0028

Table 4.10: NDCG Score comparison between BN and GNN-Bet of top-k nodes for betweenness centrality.

Dataset	BN					
	Top-100	**Top 1%**	**Top 2%**	**Top 3%**	**Top 5%**	**Top 10%**
Wiki-Vote	0.998	0.996	0.996	0.995	0.995	0.995
P2p-Gnutella31	0.984	0.975	0.972	0.972	0.971	0.970
Soc-Epinions	0.998	0.995	0.993	0.991	0.988	0.984
Soc-Slashdot	0.997	0.993	0.993	0.989	0.986	0.983
Ego-Gplus	0.998	0.994	0.991	0.990	0.987	0.983
Email-EuAll	0.993	0.987	0.986	0.985	0.986	0.986
Web-Google	0.999	0.990	0.985	0.981	0.977	0.971
Wiki-Talk	0.966	0.921	0.911	0.911	0.912	0.915
Synthetic-ER	0.852±0.1181	0.867±0.0994	0.873±0.0945	0.877±0.0908	0.881±0.0869	0.892±0.0796
Synthetic-SF	0.994±0.0022	0.986±0.0033	0.984±0.0045	0.982±0.0038	0.983±0.0035	0.988±0.0024
Synthetic-GRP	0.323±0.1489	0.342±0.1457	0.337±0.1356	0.346±0.1351	0.371±0.1411	0.392±0.1184

Dataset	GNN-Bet					
	Top-100	**Top 1%**	**Top 2%**	**Top 3%**	**Top 5%**	**Top 10%**
Wiki-Vote	0.945	0.951	0.952	0.959	0.964	0.976
P2p-Gnutella31	0.901	0.934	0.947	0.953	0.959	0.967
Soc-Epinions	0.660	0.716	0.751	0.782	0.823	0.873
Soc-Slashdot	0.764	0.862	0.900	0.919	0.934	0.950
Ego-Gplus	0.408	0.488	0.563	0.606	0.649	0.700
Email-EuAll	0.694	0.831	0.841	0.852	0.867	0.868
Web-Google	0.611	0.609	0.652	0.684	0.726	0.788
Wiki-Talk	0.605	0.755	0.765	0.770	0.782	0.807
Synthetic-ER	0.981±0.0192	0.984±0.0166	0.986±0.0151	0.987±0.0140	0.988±0.0120	0.990±0.0095
Synthetic-SF	0.935±0.0770	0.945±0.0555	0.951±0.0452	0.954±0.0409	0.959±0.0342	0.969±0.0266
Synthetic-GRP	0.981±0.0075	0.986±0.0048	0.988±0.0042	0.989±0.0040	0.992±0.0033	0.993±0.0028

Table 4.11: NDCG Score comparison between OC and GNN-Close of top-k nodes for closeness centrality.

Dataset	OC					
	Top-100	**Top 1%**	**Top 2%**	**Top 3%**	**Top 5%**	**Top 10%**
Wiki-Vote	0.986	0.983	0.988	0.987	0.990	0.994
P2p-Gnutella31	0.985	0.990	0.993	0.994	0.995	0.997
Soc-Epinions	0.992	0.995	0.996	0.996	0.997	0.999
Soc-Slashdot	0.993	0.996	0.998	0.998	0.999	0.999
Ego-Gplus	0.992	0.996	0.997	0.997	0.998	0.999
Email-EuAll	0.983	0.988	0.991	0.986	0.988	0.991
Web-Google	0.998	0.998	0.998	0.997	0.998	0.998
Wiki-Talk	0.996	0.999	0.999	0.999	0.999	1.000
Synthetic-ER	0.941±0.0587	0.970±0.0274	0.978±0.0183	0.982±0.0137	0.986±0.0096	0.991±0.0052
Synthetic-SF	0.848±0.0715	0.928±0.0401	0.950±0.0296	0.959±0.0261	0..966±0.0210	0.977±0.0125
Synthetic-GRP	0.698±0.2714	0.712±0.2687	0.746±0.2321	0.759±0.2162	0.795±0.1783	0.800±0.1886

Dataset	GNN-Close					
	Top-100	**Top 1%**	**Top 2%**	**Top 3%**	**Top 5%**	**Top 10%**
Wiki-Vote	0.972	0.977	0.976	0.973	0.978	0.983
P2p-Gnutella31	0.965	0.968	0.974	0.977	0.982	0.989
Soc-Epinions	0.921	0.939	0.953	0.960	0.969	0.979
Soc-Slashdot	0.982	0.985	0.990	0.991	0.992	0.993
Ego-Gplus	0.714	0.760	0.809	0.844	0.882	0.929
Email-EuAll	0.971	0.983	0.979	0.974	0.974	0.980
Web-Google	0.476	0.775	0.807	0.826	0.843	0.877
Wiki-Talk	0.579	0.845	0.860	0.869	0.883	0.914
Synthetic-ER	0.973±0.0232	0.979±0.0018	0.980±0.0169	0.981±0.0168	0.983±0.0161	0.985±0.0148
Synthetic-SF	0.948±0.0386	0.967±0.0143	0.970±0.0108	0.972±0.0108	0.972±0.0125	0.978±0.0075
Synthetic-GRP	0.992±0.0014	0.995±0.0009	0.995±0.0008	0.996±0.0008	0.997±0.0007	0.997±0.0007

Table 4.12: NDCG Score comparison between CD and GNN-Close of top-k nodes for closeness centrality.

Dataset	CD					
	Top-100	**Top 1%**	**Top 2%**	**Top 3%**	**Top 5%**	**Top 10%**
Wiki-Vote	0.001	0.001	0.001	0.001	0.001	0.001
P2p-Gnutella31	0.001	0.001	0.001	0.013	0.362	0.649
Soc-Epinions	0.000	0.000	0.000	0.000	0.000	0.451
Soc-Slashdot	0.000	0.000	0.000	0.000	0.000	0.132
Ego-Gplus	0.000	0.000	0.000	0.195	0.483	0.717
Email-EuAll	0.000	0.000	0.000	0.000	0.000	0.000
Web-Google	0.000	0.000	0.000	0.000	0.000	0.000
Wiki-Talk	0.000	0.175	0.561	0.692	0.802	0.896
Synthetic-ER	0.406±0.4577	0.505±0.4630	0.526±0.4733	0.546±0.4655	0.619±0.4150	0.689±0.3930
Synthetic-SF	0.001±0.0006	0.0015±0.0007	0.002±0.0007	0.002±0.0007	0.002±0.0007	0.002±0.0008
Synthetic-GRP	0.002±0.0031	0.004±0.0045	0.004±0.0054	0.005±0.0060	0.006±0.0071	0.008±0.0090

Dataset	GNN-Close					
	Top-100	**Top 1%**	**Top 2%**	**Top 3%**	**Top 5%**	**Top 10%**
Wiki-Vote	0.972	0.977	0.976	0.973	0.978	0.983
P2p-Gnutella31	0.965	0.968	0.974	0.977	0.982	0.989
Soc-Epinions	0.921	0.939	0.953	0.960	0.969	0.979
Soc-Slashdot	0.982	0.985	0.990	0.991	0.992	0.993
Ego-Gplus	0.714	0.760	0.809	0.844	0.882	0.929
Email-EuAll	0.971	0.983	0.979	0.974	0.974	0.980
Web-Google	0.476	0.775	0.807	0.826	0.843	0.877
Wiki-Talk	0.579	0.845	0.860	0.869	0.883	0.914
Synthetic-ER	0.973±0.0232	0.979±0.0018	0.980±0.0169	0.981±0.0168	0.983±0.0161	0.985±0.0148
Synthetic-SF	0.948±0.0386	0.967±0.0143	0.970±0.0108	0.972±0.0108	0.972±0.0125	0.978±0.0075
Synthetic-GRP	0.992±0.0014	0.995±0.0009	0.995±0.0008	0.996±0.0008	0.997±0.0007	0.997±0.0007

Table 4.13: NDCG Score for top-k nodes for GNN-Bet when trained on synthetic graph datasets and derived real-world graph datasets.

Dataset	Training data	GNN-Bet			
		Top-100	**Top 1%**	**Top 2%**	**Top 3%**
Wiki-Vote	Synthetic	0.945	0.951	0.952	0.959
	Derived	**0.966**	**0.971**	**0.968**	**0.972**
P2p-Gnutella31	Synthetic	0.901	0.934	0.947	0.953
	Derived	**0.908**	**0.939**	**0.951**	**0.955**
Soc-Epinions	Synthetic	0.660	0.716	0.751	0.782
	Derived	**0.877**	**0.940**	**0.956**	**0.962**
Soc-Slashdot	Synthetic	0.764	0.862	0.900	0.919
	Derived	**0.920**	**0.965**	**0.973**	**0.976**
Ego-Gplus	Synthetic	0.408	0.488	0.563	0.606
	Derived	**0.866**	**0.870**	**0.888**	**0.894**
Email-EuAll	Synthetic	0.694	0.831	0.841	0.852
	Derived	**0.846**	**0.941**	**0.946**	**0.954**
Web-Google	Synthetic	**0.611**	0.609	0.652	0.684
	Derived	0.546	**0.760**	**0.796**	**0.816**
Wiki-Talk	Synthetic	0.605	0.755	0.765	0.770
	Derived	**0.857**	**0.907**	**0.908**	**0.909**

other neural network based methods, they do require some hyper-parameters that need to be set to get the best out of the model. With ablation experiments, we see that the model's performance is relatively stable with various combinations of parameters. Thus it does not require extensive hyper-parameter tuning. In Appendix A.2, we briefly discuss about oversmoothing in GNNs and our observations with respect to the proposed model. For betweenness and closeness approximations, we have set the number of layers in our model to four and seven respectively for all graph datasets to compare results on a single model. Nevertheless, the number of layers can be chosen based on the structure and diameter of the target graph.

Even though the models are trained on synthetic scale-free graphs, tests on real-world graph datasets show good approximation accuracy. This implies that the proposed model is inductive in nature and has the ability to learn on one set of graphs and predict node ranking scores on others. Furthermore, the model is invariant to the node sequence in the adjacency matrix, providing the additional benefit of having no constraint of ordering the nodes to any specific sequence in graphs.

With almost all deep learning models, even though training time may vary, usually their inference time is really small. Another advantage of neural network models is to take advantage of high-performance GPUs for faster calculations. Our proposed model has the advantage of small training time as well as inference time.

As shown in results, our proposed frameworks are able to perform really well on different types of graphs. However, since the model training is based on ranking loss function where gradients rely on differences in score values. Therefore, it is possible that if the given graph has lots of nodes with similar (or same) centrality values, the model may not be able to distinguish the ranking of nodes and may not perform very well.

4.5 Related Work

In previous sections, we have shown the viability of neural network based approach for approximating betweenness and closeness centrality of nodes in the graphs. Our proposed model provides a new way to approach the problem and adds to the toolset in addition to algorithmic based approaches. In this section, we provide a brief summary of current related works on betweenness and closeness centrality approximation methods and various applications of neural networks on graph-structured data.

4.5.1 Betweenness Centrality

Intuitively, we have to find all the shortest paths between all pairs of nodes to calculate betweenness. Brandes[Brandes 2001] improves upon the naive algorithm by proposing to generate directed acyclic graph (DAG) starting from each node $v \in V$ using BFS and aggregating the contributions of nodes on paths in DAG for all nodes. Calculating exact betweenness centrality value for nodes is computationally expensive and has a time complexity of $\Theta(|V||E|)$. Hence it is prohibitive for large graphs. Two different approaches have been taken to tackle the problem of high computation time. The first approach is to adapt the betweenness centrality algorithm for distributed computation. Recently many algorithms[Edmonds, Hoefler, and Lumsdaine 2010; Hoang et al. 2019; Crescenzi, Fraigniaud, and Paz 2020] have been proposed which extend the computation from a single machine to a cluster of high performance machines. This approach reduces the computation time significantly for large graphs. However, access to such computation resources is rather limited. The second approach is to rely on approximation methods.

In most of the cases, we do not require an exact calculation of rankings of nodes, so an approximation is sufficient for tasks in hand. Many algorithms have been proposed. Brandes and Pich [Brandes and Pich 2007] proposed an algorithm by taking a subset of k starting nodes (pivots) rather than all the nodes. Selecting these pivots uniformly at random and with $k << N$ where N is the number of nodes

and betweenness centrality values are extrapolated in the graph. Unfortunately, this method produces large overestimates for unimportant nodes that happen to be near the pivot. Geisberger et al. [Geisberger, Sanders, and Schultes 2008] solve the problem of overestimation by changing the scheme of aggregation of betweenness contributions so that the nodes near pivot do not get extra contributions. To limit the bias, they introduce a scaling function, which gives less importance to contributions from the pivots that are close to the node. Since this algorithm works with random sampling of nodes (pivots), as the number of samples increases, approximation accuracy also increases. Although a higher number of samples also cause higher computation overhead of calculating more shortest paths in the graph.

Riondato et al.[Riondato and Kornaropoulos 2016] takes a different approach while providing a theoretical guarantee. The idea is to sample a set of r shortest paths between randomly sampled source-target pairs (s,t). Then the algorithm computes the approximated betweenness centrality of a given node v as the fraction of sampled paths that v is internal to, by adding $\frac{1}{r}$ to the node's score for each of these paths. Borassi et al.[Borassi and Natale 2019] provide a new adaptive sampling technique for sampling shortest paths, which provides a faster computation of betweenness within a given absolute error.

While finalizing this manuscript, we became aware that Fan et al. [Fan et al. 2019a] have also concurrently proposed a GNN based method for approximation of betweenness centrality. There are several significant differences from our work. First, their method is limited to undirected graphs, while ours is proposed for the more general case of directed graphs. Second, their method is limited to betweenness centrality in static graphs while our model is more flexible and is able to handle both betweenness and closeness centrality in both static and dynamic graphs. Third, their method is proposed for identifying top-N high betweenness nodes while our model is able to make a complete ranking of all the nodes in a graph. Due to these constraints, we do not compare to this work[Fan et al. 2019a] in our paper.

In addition to the regular graphs, the concept of betweenness centrality has been applied to directed hypergraphs[Ausiello and Laura 2017], where relations (also called hyper-edges) are defined from a set of source nodes to a single target node. [Lee and Kim 2020] proposes a new measure called Participation-based Between Centrality (PBC) for a special case of directed hypergraphs called B-hypergraphs and provides an exact and approximate method for calculation of PBC.

4.5.2 Closeness Centrality

To approximate the closeness centrality Eppstein et al.[Eppstein and Wang 2001] and Okamoto et al.[Okamoto, Chen, and Li 2008] proposed algorithm based on sampling k nodes from the graph. However, these algorithms are not optimal when distance distribution of sampled nodes with other nodes is heavy-tailed. Cohen et al. [Cohen et al. 2014] provided an improved algorithm with a hybrid approach of sampling and pivoting. For directed graphs, there are two variations of closeness centrality based on the direction of paths. One is incoming closeness centrality for paths incoming to the node and outgoing closeness centrality for paths going away from the given node to other nodes. In this thesis, we have focused on outgoing closeness centrality as it is more meaningful in terms of node importance to spread information.

4.5.3 Centrality for Dynamic Graphs

In the real world, node interactions and the level of activity of nodes change all the time. Some nodes and edges can disappear and new nodes and edges can appear with time. There is some research on updating betweenness centrality of nodes in dynamic graphs. A naive way is to recompute from scratch, which is however computationally expensive. Many dynamic algorithms[Green, McColl, and Bader 2012; Kas et al. 2013; Lee et al. 2012; Kourtellis, Morales, and Bonchi 2016] have been proposed which avoid recomputation by maintaining data structure based on original graph structure and update it as the structure changes. These algorithms have faster updates compared to Brandes's exact algorithm. However, they are not scalable as they require a huge amount of memory $\sigma(n^2)$ to store the data structures.

Bergamini et al. [Bergamini, Meyerhenke, and Staudt 2014] proposed a method to approximate the dynamic centrality of nodes with the insertion of edges. This method is based on Riondato et al.[Riondato and Kornaropoulos 2016], although the proposed method does not support the removal of edges. Hayashi et al.[Hayashi, Akiba, and Yoshida 2015] propose a new method based on the data structure named *hypergraph sketch*, which is used to represent the shortest paths between nodes sampled randomly. Their proposed method supports both nodes and edges addition/removal operations.

4.5.4 Graph Neural Networks

Due to an abundance of graph-structured data, many graph-centric neural network models have been proposed. These GNNs have been designed to solve multitude of problems like node classification[Kipf and Welling 2017; Velickovic et al. 2018b;

Velickovic et al. 2018a], graph classification[Ying et al. 2018b][Zhang et al. 2018], link prediction[Ying et al. 2018a; Berg, Kipf, and Welling 2017], prediction of molecular properties[Gilmer et al. 2017], functional analysis of brain[Parisot et al. 2017; Rakhimberdina and Murata 2020], synthesize chemical compounds[Li et al. 2018; Cao and Kipf 2018] and so on.

All of these GNN frameworks utilize node or edge feature aggregation scheme guided by the structure of the graph[Wu et al. 2019b; Zhou et al. 2019]. In addition to the simple summation for node features as described in this paper, Hamilton et al.[Hamilton, Ying, and Leskovec 2017] have proposed aggregation based on LSTM and max-pooling operation of randomly sampled neighbors. Earlier proposed GNN models were transductive in nature, in other words the models can only be trained and tested on the same graph[Kipf and Welling 2017; Defferrard, Bresson, and Vandergheynst 2016; Defferrard, Bresson, and Vandergheynst 2016]. However, there have been many emerging works, which are inductive by construction[Hamilton, Ying, and Leskovec 2017; Chen, Ma, and Xiao 2018; Velickovic et al. 2018a]. The models can be trained on one set of graphs and tested on another set of graphs. Recent works are now delving deeper into the working of GNNs from a theoretical perspective, exploring their learning capacity[Corso et al. 2020] and expressiveness with respect to graph isomorphism heuristic algorithms[Xu et al. 2019; Maron et al. 2019; Morris et al. 2018].

4.6 Conclusion

In this paper, we propose a novel approach to solve the betweenness and closeness approximation problem in graphs using GNN. Using the constrained message passing scheme, we train graph neural network model to output ranking scores for nodes in correlation with betweenness and closeness centrality values. We compare the performance of the model with other state of the art approximation techniques and the experimental results show better performance of our model while taking significantly less time.

For future works, possibilities can be explored to extend the neural network framework to other centrality measures. Graphs based on real-world data often accompany explicit node/edge feature information. Current graph centrality measures lack in utilizing this additional information in ranking nodes as they are solely based on the structure of the graph. However, GNN based models can easily incorporate any additional data for learning. There is a potential to find new ways to rank nodes based on the graph structure of data and node/edge external features, which explains real-world interactions more consistently.

Chapter 5

Node Classification

5.1 Introduction

One of the popular applications of GNN is the node classification task, where a GNN model learns to predict unknown labels of the nodes based on similarity or dissimilarity to the nodes for which labels are known. Early GNN models such as Graph Convolutional Network (GCN) [Kipf and Welling 2017] use simple feature aggregation schemes. Since then, researchers have incorporated many techniques from deep learning literature and proposed numerous GNN variants [Wu et al. 2019b] to address various shortcomings in GNN model training and to improve the prediction capabilities. Some of the techniques used in these variants include neighbor sampling [Hamilton, Ying, and Leskovec 2017; Chen, Ma, and Xiao 2018], attention mechanism [Velickovic et al. 2018b], use of Personalized PageRank matrix instead of adjacency matrix [Klicpera, Bojchevski, and Günnemann 2018], leveraging proximity in feature space [Jin et al. 2021] and simplified model design [Wu et al. 2019a]. Also, there has been an increasing trend in making the models deeper by stacking more layers and using the residual connections to improve the expressiveness of the model [Rong et al. 2020; Chen et al. 2020]. For example, GCNII [Chen et al. 2020] incorporates up to 64 layers with scaled residual weights. However, most of these models by design are more suitable for homophily datasets, where nodes that are linked to each other are more likely to belong to the same class. As a result, these GNNs may not perform well with heterophily datasets, which are more likely to have nodes with different labels connected together. This problem was highlighted by Zhu et al. [Zhu et al. 2020] and the authors proposed node's ego-embedding and neighbor-embedding separation to improve performance on heterophily datasets. Other recent works approach this problem in different manner e.g, [Zhu et al. 2021] utilizes belief propagation, [Bo

et al. 2021] uses adaptive gating on edges etc.

With increasing complexity in GNN architecture design, it becomes challenging to analyze GNN models and determine which components contribute more to the model's performance over a specific prediction task. In this work, we study the problem of node classification using GNN and approach it using feature importance and feature selection perspective. With results from extensive experiments, we show that input features often contain noisy components, and minimizing their effects during learning can improve prediction performance even with a simple neural network model. This also highlights that with simple feature-centric approach, use of complex and deep GNN models is often unnecessary. Based on our analysis, we propose a simple GNN model to improve prediction accuracy in the node classification task.

Our Contributions

- We run extensive experiments on multiple node classification benchmark datasets and show that aggregated features generated in GNNs have noisy components. In addition, feature selection is an essential requirement for higher prediction accuracy.
- We experimentally confirm the feature generation requirements for homophily and heterophily graphs.
- We propose a simple 2-layered GNN model, "FSGNN", which incorporates a "soft-selection" mechanism to learn the importance of features during training of the model.
- Our proposed model empirically outperforms other state-of-art (SOTA) GNN models (both shallow and deep) and achieves up to 51.1% higher node classification accuracy.

The rest of the chapter is organized as follows: Section 5.2 outlines node classification task and formulation of graph neural networks. In Section 5.3, we explore the importance of features by running extensive experiments and present our observations. Based on our experimental observations, in Section 5.4, we propose our GNN model FSGNN. In Section 5.5, we briefly introduce relevant GNN literature. Section 5.6 contains the experimental details and comparison with other GNN models. In Section 5.7, we present our results and empirically analyze our proposed design strategies and their effect on the model's performance. Section 5.8 summarizes the paper.

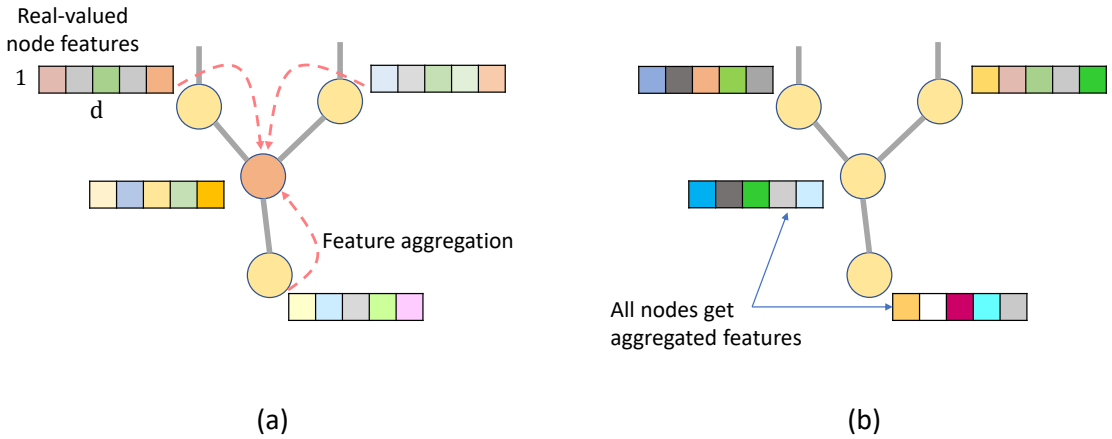


Figure 5.1: (a) Each node has a d-dimensional feature vector and aggregates the features from its neighbors. (b) After aggregation step, each node gets an updated aggregated feature vector with neighborhood information.

5.2 Preliminaries

5.2.1 Node Classification

The rapid advancement of technologies has enabled us to capture and store a tremendous amount of data. However, often this data is noisy and incomplete. For example, in social networks, users may not provide all profile related information like their geographical location, hobbies, interests, political inclination etc. These data attributes can be considered as labels associated with the users [Bhagat, Cormode, and Muthukrishnan 2011]. The labels for users can be described in many ways, for example, labels based on gender, interests, hobbies, locations, etc. By predicting these labels, online platforms can help users by suggesting them new friends with similar profiles or recommending them new products through advertisement systems. Considering a social network as a graph with nodes as users and friendship between users as edges, only a subset of nodes are labeled. Node classification is the method to classify/group all nodes based on their likely labels.

Definition 1. *For given graph $G = (V, E)$ with subset of nodes $V_\ell \subset V$ that are labeled, and $V_u = V \backslash V_\ell$ is the set of unlabeled nodes. Let Y be the set of m possible labels, and $Y_\ell = \{y_1, y_2, \cdots y_\ell\}$ be the known labels of nodes of V_ℓ. Node classification is the task to infer labels Y for all nodes in the graph.*

One approach to solve this problem is to utilize human experts to label the

nodes. However, this approach does not scale with large-sized graphs with hundreds of thousands to millions of nodes. Hence, researchers have approached this problem using computational based learning approaches. These approaches take advantage of the fact that nodes with the same labels would often share common attributes.

Learning on graph data often requires transforming the data into appropriate formats. Graph is represented as an adjacency matrix or a laplacian matrix. Attribute information pertaining to the nodes or edges is often stored in a real-valued vector representation. For example, for a node representing a user in a social network, the attributes from the user's profile are encoded in numeric form. Each index of the vector represents an individual attribute. Each node has a d-dimensional vector representation, also known as node feature vector, encoding all node attribute information (Figure 5.1). $X \in \mathbb{R}^{n \times d}$ is node feature matrix containing feature vectors of all nodes. For the node classification problem, A and X are main input for learning algorithms.

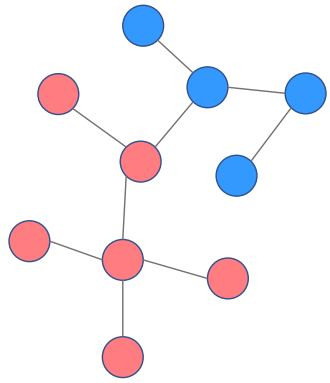
5.2.2 Homophily and Heterophily

Homophily or heterophily in a graph describes similarity or dissimilarity of attributes of connected nodes [McPherson, Smith-Lovin, and Cook 2001; Zhu et al. 2020]. A graph is said to have high homophily when most edges connect nodes with similar attributes or same labels. Conversely, a graph has high heterophily if most edges connect nodes with dissimilar attributes or different labels (Figure 5.2). One example of homophily is social networks where users with common attributes like geography, interests, political affiliations etc. are friends with each other. An example of heterophily in graphs is some dating networks where a user connects to other users of the opposite gender.

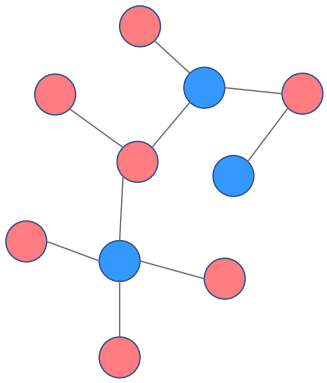
One way to measure homophily/heterophily in graph is through edge homophily ratio ($\hbar$) [Zhu et al. 2020; Zhu et al. 2021]. It is defined as a fraction of the number of edges with both nodes having same labels to the total number of edges in the graph.

$$\hbar = \frac{|\{(v_i, v_j) : (v_i, v_j) \in E \wedge y_i = y_j\}|}{|E|} \tag{5.1}$$

Graphs with strong homophily has high edge homophily ratio, $\hbar \to 1$, while graphs with strong heterophily has small edge homophily ratio, $\hbar \to 0$. Real-world datasets often show varying level of edge homophily ratio. Table 5.2 shows homophily ratio of some standard graph datasets.



Graph with strong homophily



Graph with strong heterophily

Figure 5.2: Graph with strong homophily have nodes with similar attributes or labels connected together. In case of graphs with strong heterophily, nodes with dissimilar attributes or labels are connected to each other.

As the node classification task requires identifying similarities between node attributes, the study of homophily and heterophily properties in the data helps to guide the design of the GNNs. In the next section, we provide more details on the basic architecture design of the GNN model.

5.2.3 Graph Neural Networks

Graph Neural Networks (GNNs) are a class of neural network models specifically designed to learn on graph-structured data. GNNs learn to map high dimensional graph data to low dimensional node representations, which are then used for predictions on various downstream tasks. Nodes with similar input features are embedded closer to each other in the embedding vector space, while dissimilar nodes have a greater distance between them (Figure 5.4).

Hop-Feature Generation

One of the primary operations of GNN is feature aggregation. In the feature aggregation step, each node combines the features of its neighbors (Figure 5.1). The number of aggregation steps corresponds to the number of hops a neighbor node is from the source node for feature aggregation. For example, three aggregation steps correspond

to the node aggregating features of neighboring nodes lying at up to 3-hop distance from the node in the graph. Sharing of feature information among neighbors helps to define common neighborhood properties, which in turn helps GNN models to learn and predict effectively.

Feature aggregation step is calculated by matrix multiplication of adjacency matrix with node feature matrix. For K-step aggregation, aggregated features are calculated as $A^k X \in \mathbb{R}^{n \times d}, k \in (1 \cdots K)$. Hence, with multiple aggregation steps, each node gets an additional feature information vector corresponding to the neighborhood at each hop.

Layer design

GNNs leverage feature propagation mechanism [Gilmer et al. 2017] to aggregate neighborhood information of a node and use non-linear transformation with trainable weight matrix to get the final embeddings for the nodes. Conventionally, a simple GNN layer is defined as

$$H^{(i+1)} = \sigma(\tilde{A}_{sym} H^{(i)} W^{(i)}) \tag{5.2}$$

where $\tilde{A}_{sym} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ is a symmetric normalized adjacency matrix with added self-loops. $H^{(i)}$ represents features from the previous layer, $W^{(i)}$ denotes the learnable weight matrix, and σ is a non-linear activation function, which is usually ReLU in most implementations of GNNs. However, this formulation is suitable for homophily datasets as features are cumulatively aggregated, i.e. node's own features are added together with neighbor's features. The cumulative aggregation of node's self-features with that of neighbors reinforces the signal corresponding to the label and helps to improve accuracy of the predictions. On the other hand, in the case of heterophily, nodes are assumed to have dissimilar features and labels to their neighbors. For heterophily datasets, H2GCN [Zhu et al. 2020] proposes to separate features of neighbors from node's own features, thus avoiding aggregation of dissimilar features. So we use the following formulation for the GNN layer,

$$H^{(i+1)} = \sigma(A_{sym} H^{(i)} W^{(i)}) \tag{5.3}$$

where $A_{sym} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is symmetric normalized adjacency matrix without added self-loops. Figure 5.3 shows difference between no-loop and self-looped aggregation. To combine features from multiple hops, concatenation operator can be used before the final layer.

One of the popular example of GNN is Graph Convolutional Network (GCN) [Kipf and Welling 2017]. GCN is a two-layered GNN model with layer design based on Equation 5.2. Formulation of GCN model is defined as,

$$Z = \tilde{A}_{sym}\sigma(\tilde{A}_{sym}XW^{(0)})W^{(1)} \tag{5.4}$$

GCN aggregates the features of two hops of neighbors and outputs Z. Softmax function is applied row-wise , and cross-entropy error is calculated over all labeled training examples. The gradients of loss are back-propagated through the GNN layers. Once trained, the model can be used to predict labels of the nodes in the test set.

5.2.4 Node Classification with GNNs

Node classification is an extensively studied graph based semi-supervised learning problem. It encompasses training the GNN to predict unknown labels of nodes based on the features and neighborhood structure of the nodes. GNN model is considered as a function $f(X, A)$ conditioned on node features X and adjacency matrix A. After training the model, nodes with similar features and proximity in graph structure are embedded closer to each other in vector space forming clusters as shown in Figure 5.4. Then, unknown labels of the nodes can be predicted based on distance from labeled nodes in vector or embedding space.

For clarity, we provide an example of node classification using Cora, a real-world benchmark dataset for node classification.

Example: *Cora is a citation graph for machine learning papers. It has 2,708 nodes and 5,429 edges.*

Nodes V: Each academic paper is denoted as a node.

Edges E: An edge is defined when a paper cites another paper.

Node Labels Y: The papers are divided into seven classes based on the research field of these papers. Labels for nodes in the training set are visible to the GNN model. Once GNN is trained, predicted labels of the test set are compared with real labels.

Node Features X: Node feature vector is bag of word representation of 1,433 distinct words [Sen et al. 2008; Zhu et al. 2020]. Each index contains 1 or 0 depending upon the presence or absence of the word in the paper. Papers in the same field may share common words related to the area. Hence, the nodes representing them will have similar feature vectors.

In order to train a GNN model, the dataset is split into training, validation, and test set. Labels of nodes in the training set are accessible to the GNN model during training. Validation set is used to tune the hyper-parameters of the model. Once the model is trained, labels of nodes in the test set are predicted using the model. Predicted labels are compared with the real labels of the nodes and the accuracy

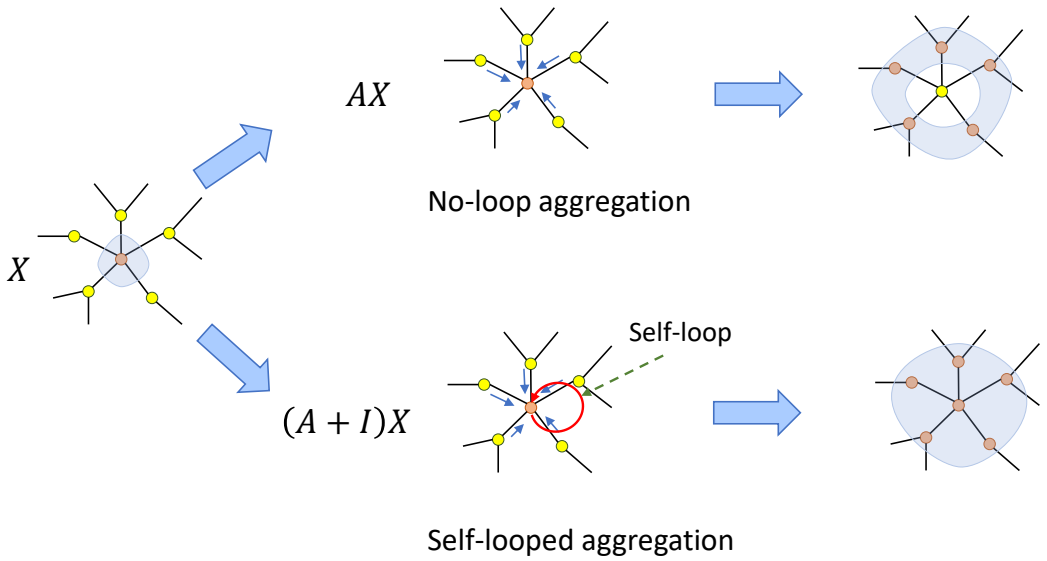


Figure 5.3: No-loop (top) and Self-loop (bottom) aggregation of node features. No-loop aggregation step does not combine the node's features to neighbors' as they are dissimilar (heterophily). Self-looped aggregation step combines the node's features with neighbors' as they share common attributes (homophily).

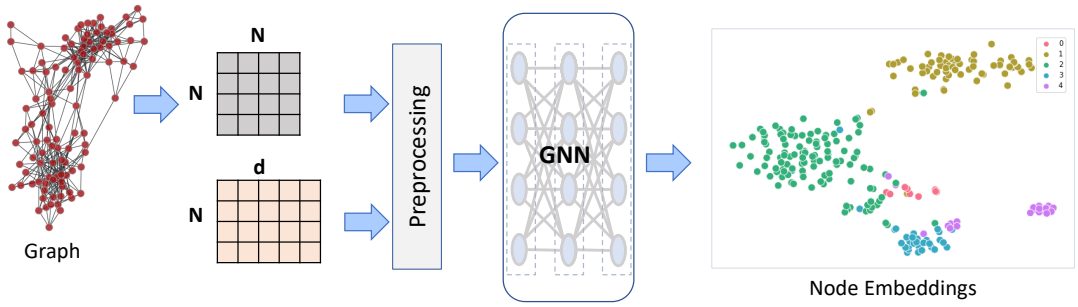


Figure 5.4: Figure shows a simple representation of graph neural network. Graph is represented as adjacency matrix and along with node feature matrix is set as input to GNN model. Preprocessing includes normalization of the adjacency matrix and the node feature matrix. GNN models learns to map input node features to node representations in embedding space. Colors of the nodes denote their class labels.

of prediction is calculated. Figure 5.5 shows the embedding plots of original node features and learned node embeddings after training a GNN model (FSGNN) on the Cora dataset. For initial node features (on left), we observe that distance between intra-class nodes and inter-class nodes is not significant. On the other hand, for learned node features, the inter-class distance is significantly higher and nodes of same class form higher quality clusters. Thus, the model can predict unknown labels of the nodes with higher accuracy.

5.2.5 Feature Generation in GNNs

As discussed in section 5.2.3, we can generate different features for the nodes capturing homophilic and heterophilic properties in the graph by modifying the feature aggregation step. The number of different features generated is dependent on the number of hops of aggregation over neighbors. In addition, node features can also be generated based on proximity in feature space or based on some other arbitrary criterion. Feature generation steps can differ among GNNs based on their architecture. Many GNN models have feature aggregation and representation learning combined in single layer [Kipf and Welling 2017; Klicpera, Bojchevski, and Günnemann 2018; Chen et al. 2020], while in other models features can be precomputed beforehand [Wu et al. 2019a; Frasca et al. 2020].

Figure 5.5: Figure shows t-SNE embedding plots of the node features of Cora dataset. Each node belongs to one of the seven classes represented by different colors. (a) Plot of initial node features, (b) Plot of node embeddings learned after training GNN model on the dataset.

5.2.6 Challenges with current GNNs

Following the progress in the deep learning field, there have been many recent advances in the design of GNNs. Recent works have proposed to make models deeper, using residual connections, attention layers, scaling output of GNN layers and residual connections etc. However, these improvements also bring challenges of increased complexity in GNN models. The new GNN models requires more number of hyperparameters that need to be tuned. Moreover, it also becomes difficult to analyze which components of the model are contributing to the performance in prediction tasks. These challenges make it harder for any practitioner to decide which GNN model or architecture to use out of many when approaching a new problem or task.

In the next section, we reevaluate the problem of node classification and try to simplify the approach to GNN model design. We explore the problem from the perspective of feature selection and demonstrate that we can reduce the complexity of the GNN model by using an appropriate feature selection strategy.

5.3 Feature Selection in Graph Neural Networks

On any given graph-structured data, a set of features can be generated for the nodes (e.g. using Eq. (5.2) & (5.3)). The number of features depends on the problem in hand, properties of the dataset, design choice of practitioner etc.

We assume a function,

$$g(X, A, K) \mapsto \{X_1, X_2, \ldots, X_l\}$$

The function takes $X \in \mathbb{R}^{n \times d}$ as node features matrix, A as an adjacency matrix, K as the power of the adjacency matrix or number of hops to propagate features and outputs a set of l node features.

With the increase in value of K, the node aggregates features of other nodes that lie multiple hops away from it. Features of such nodes may not be similar or correlated with that of the source node. Thus, some aggregated features may be uninformative for learning over a task. Finding the best value of K can require some experimental exploration, and often practitioner chooses K based on the task and dataset.

In the node classification task, for given label distribution, only a subset of these features are relevant to predict the label of the node. For example, a feature X_i is relevant to class C_i, if X_i and C_i are highly correlated[Tang, Alelyani, and Liu 2014; Li et al. 2017a; Chandrashekar and Sahin 2014]. Irrelevant or noisy features may not correlate with target labels but can still affect the learning process.

In this section, we explore the importance of features generated by the aggregation step at different hops. We run a series of extensive experiments to study how different features affect predictions for graph neural networks in the node classification task. Using these experiments, we aim to answer the following three questions:

Q.1 *How useful are individual features generated from multi-step aggregation in graph neural networks?*

Q.2 *What is the effect of training the model over all the features, and what are the effects of different aggregator schemes?*

Q.3 *What is the impact of adding or removing features on the model's performance?*

Exploring these questions provides a deeper understanding of how GNN models can be designed to have better prediction capabilities.

5.3.1 Experiment Setting

Model Design

Conventionally, GNN models have feature propagation and transformation combined into a single layer, and the layers are stacked together. This step makes it difficult to distinguish the importance of the features and the role of MLP, and it becomes harder to analyze their impact on the prediction results of the model. For our experiments,

we decouple the feature generation step and representation learning over features separately.

We use 2-layer neural network for our experiments. For the model design, instead of a single input channel, we propose to have all these features as input in parallel. Each feature is mapped to a separate linear layer. Hence the linear transformations are uniquely learned for all input features. In addition, we use L2-normalization to row-wise normalize the output of the linear layer. L2-normalization scales the node embedding vectors to lie on the "unit sphere". `ReLU` and `Dropout` are used for non-linear transformation of hidden features and regularization respectively. In the case of a single input feature matrix, hidden features are mapped to the final layer, and in the case of multiple input features, all hidden features of the node are aggregated and mapped to the final layer. We use two different aggregator schemes: `sum` and `concatenation`, and compare the results.

Input Features

In our experiments, we consider node features of up to 3-hop neighbors (commonly used setting) in the graph. As we analyze both heterophily and homophily properties in a graph, we calculate both self-looped and no-loop features of the nodes. Hence, including node's own features, our feature set has total of 7 (1+2×3) different features for the node $X_{feat} = \{X, AX, (A+I)X, A^2X, (A+I)^2X, A^3X, (A+I)^3X\}$. To explore the answers to the three questions defined earlier, we design three experiment settings: `Single_feature`, `All_feature` and `Sub_feature`. In `Single_feature` setting, we use only one out of seven features to train the model, and results are compared among all features. In this way, we analyze how informative each feature is for the label prediction of the nodes. In `All_feature` setting, we train the model on all features together and ascertain the model's performance. In addition, we use two aggregation schemes, i.e. sum and concatenation of hidden node features as they are commonly used in GNN models. Please note that `All_feature` with concatenate setting is similar to the SIGN model[Frasca et al. 2020] as the model uses simple concatenation of features. In `Sub_feature` setting, we train the model on all possible combinations of the features in X_{feat} excluding the subsets already used in `Single_feature` and `All_feature`. Examples of such subset are $\{X, A^2X, (A+I)^3X\}$ and $\{X, AX\}$. Hence in this setting, we train the model on 119 different subsets of input features. Using all combinations of feature matrices, we perform an explicit selection of feature matrices. We report the result of the subset with the best accuracy. We use both aggregator schemes in this case too.

Datasets and Hyperparameters

We run experiments on nine different datasets with varying homophily and heterophily properties. More details on datasets and preprocessing are provided in Section 5.6. For each input feature setting, we perform a search over 54 hyperparameter combinations of learning rate, weight decay and dropout.

5.3.2 Analysis of results

Table 5.1 shows the mean node classification accuracy with different input features for hidden dimension size of 64. We also include current state-of-the-art (SOTA) results for a comparison with our results. We find many interesting observations as follows:

Observation 1. We find that each hop features contribute differently to the prediction performance of the model. Some hop features are more informative than the others. For homophily datasets: Cora, Citeseer, and Pubmed self-looped features have higher node classification accuracy. In many recent publications that have considered heterophily, there is often more emphasis on Texas, Wisconsin and Cornell as good heterophily datasets, and Squirrel and Chameleon are considered to have low-quality node features as heterophily datasets [Zhu et al. 2020]. However, we observe that for Wisconsin, Texas, Cornell and Actor, the best features are node's own features, and features of the neighbors are not informative enough. In case of Squirrel and Chameleon, we achieve best performance with node's first hop no-loop features. In these two datasets, node's own features and self-looped features have a low correlation with node's labels. Hence, they are, in fact, very good representation of heterophily datasets. These observations highlight the importance of using both self-looped and no-looped adjacency matrices in GNNs for better generalization over homophily and heterophily datasets respectively.

Based on our above observations, we postulate that *the problem of homophily and heterophily in graphs is a feature generation problem.* With appropriate feature generation measures, GNNs can learn on different types of graph datasets.

Observation 2. In `All_feature` setting, we train the model on all features with both concatenation and sum as aggregation operation. Our first observation is improved performance compared to `Single_feature`, which is natural as all features in combination provide more information. Comparing the aggregator schemes, for many datasets: Chameleon, Wisconsin, Texas, Cornell, and Squirrel sum operation has significantly lower accuracy compared to concatenation operation even with higher dimension embeddings (Please refer to Table B.1 for additional results with d=128 & 256). In this setting, we find concatenation operation overall provides

Table 5.1: Mean Classification Accuracy on fully-supervised node classification task on 2-layered MLP with hidden dimension size as 64. **CAT** and **SUM** refers to concatenation and sum aggregation operation respectively.

Dataset	Single_feature							All_feature		Sub_feature		SOTA
	$\mathbf{X}$	$\mathbf{AX}$	$(\mathbf{A}+\mathbf{I})\mathbf{X}$	$\mathbf{A^2X}$	$(\mathbf{A}+\mathbf{I})^2\mathbf{X}$	$\mathbf{A^3X}$	$(\mathbf{A}+\mathbf{I})^3\mathbf{X}$	CAT	SUM	CAT	SUM	
Cora	73.40	79.55	84.28	83.86	85.47	83.58	85.41	87.68	87.5	**88.10**	88.04	88.49 [Chien et al. 2021; Chen et al. 2020]
Citeseer	71.66	69.10	73.53	72.38	74.07	70.55	73.92	77.08	77.09	**77.52**	77.43	77.99 [Pei et al. 2020]
Pubmed	87.79	81.77	88.27	84.70	88.06	83.06	86.63	89.75	89.55	**89.88**	89.83	90.30 [Chen et al. 2020]
Chameleon	46.05	77.74	71.22	76.07	71.77	75.26	71.62	75.61	72.25	**78.59**	78.55	66.47 [Chien et al. 2021]
Wisconsin	87.45	63.13	58.03	62.54	52.94	60.00	51.76	85.09	79.8	87.84	**88.62**	86.98 [Suresh et al. 2021]
Texas	85.40	66.21	61.35	67.29	58.64	62.43	58.10	84.32	78.91	88.64	**88.91**	86.49 [Chien et al. 2021]
Cornell	85.94	58.64	63.51	58.64	61.62	58.91	60.27	81.89	72.25	86.21	**86.75**	82.16 [Zhu et al. 2020]
Squirrel	30.24	73.18	63.79	71.28	63.37	64.42	62.82	73.02	64.68	**74.16**	73.12	49.03 [Chien et al. 2021]
Actor	35.32	25.47	29.22	25.38	27.95	25.27	26.43	35.15	35.39	35.63	**35.67**	36.53 [Suresh et al. 2021]

better accuracy values compared with sum operation.

Observation 3. In `Sub_feature` setting, we find significant improvements in performance of the model on all datasets compared to both `Single_feature` setting and `All_feature` settings. This observation implies that among all features, there are subsets of features that are more informative than others for better prediction performance. In addition, other less informative features, if present in the input, can act as noise and may lead to worse performance of the model. This leads to the idea that feature selection is an important aspect of the design of graph neural networks. By reducing the effect of less informative/noisy features, higher prediction accuracy can be achieved even with a simple two-layered neural network. Over-smoothing problem in GNNs is considered to be due to node features becoming less informative because of feature averaging over many hops. However, with a feature selection mechanism, these uninformative features can be ignored. With these observations, we postulate that *graph learning and over-smoothing mitigation is a feature selection problem.* With a good feature selection strategy, GNN model can provide good prediction accuracy and eliminate the over-smoothing problem.

In addition, we find another interesting observation that in `Sub_feature` setting, the difference in the performance of concatenation and sum aggregation operation is reduced and is not as significant as observed with `All_feature` setting.

5.4 Proposed Architecture

As discussed in Section 5.3.2 that feature selection is important to improve prediction capability of the model. However, using `Sub_Feature` is not feasible for real-world applications as the number of input feature combinations increase exponentially with an increase in the number of hops, making it computationally expensive. Nevertheless, a GNN model can be designed to approximate feature selection strategy. When all input features are provided, the model should be able learn to assign higher weights to more relevant and informative features and actively reject features that are not useful. To construct such a model, we propose to weight input features with a single scalar value that is multiplied to each input feature matrix. We impose a constraint on these scalar values by the softmax function as follows. Let α_i be the scalar value for the i^{th} feature matrix, then α_i scales the magnitude of the features as $\alpha_i X_i W_i^{(0)}$. Softmax function is used in deep learning as a non-linear normalizer, and its output is often practically interpreted as probabilities. Before training, the scalar values corresponding to each feature matrix are initialized with equal values, and softmax is applied on these values. The resultant normalized values α_i are then multiplied with the input features, and the concatenation operator is applied. Con-

sidering L number of input feature matrices X_l, $l \in \{1 \,..\, L\}$, the formulation can be described as,

$$H^{(1)} = \Big\|_{l=1}^{L} \alpha_l X_l W_l^{(0)} \tag{5.5}$$

where $\sum_{l=1}^{L} \alpha_l = 1$ and $\|$ denotes concatenation operation.

While training, the scalar values of relevant features corresponding to the labels increase towards 1 while others decrease towards 0. The features that are not useful and represent more noise than signal have their magnitudes reduced with a corresponding decrease in their scalar values. Since we are not using a binary selection of features, we term this selection procedure as "soft-selection" of features.

The formulation discussed above can be understood in two ways. As GNNs have been represented with a polynomial filter,

$$g_\theta(P) = \sum_{k=0}^{K-1} \theta_k P^k \tag{5.6}$$

where $\theta \in \mathbb{R}^K$ is a vector of polynomial coefficients and P can be adjacency matrix [Kipf and Welling 2017; Chen et al. 2020; Chien et al. 2021], laplacian matrix [NT, Maehara, and Murata 2020] or PageRank based matrix [Berberidis, Nikolakopoulos, and Giannakis 2019]. As the polynomial coefficients are scalar parameters then our scheme can be considered as applying regularization on these parameters using the softmax function. The other way to look is to simply consider it as a weighting scheme. The input features can be arbitrarily chosen, and instead of a scalar weighting scheme, a more sophisticated scheme can be used.

For practical implementation since all weights are initialized as equal, they are all set to 1. After normalizing with softmax function, the individual scalar values become equal to $1/L$. During training, these values change, denoting the importance of the features. As the scalar values affect the magnitude of the features, they also affect the gradients propagated back to the linear layer, which transforms the input features. Hence it is important to have a unique weight matrix for each input feature matrix.

Feature Selection Graph Neural Network

Combining the model designs formulated earlier, we propose a simple and shallow (2-layered) graph GNN model called Feature Selection Graph Neural Network

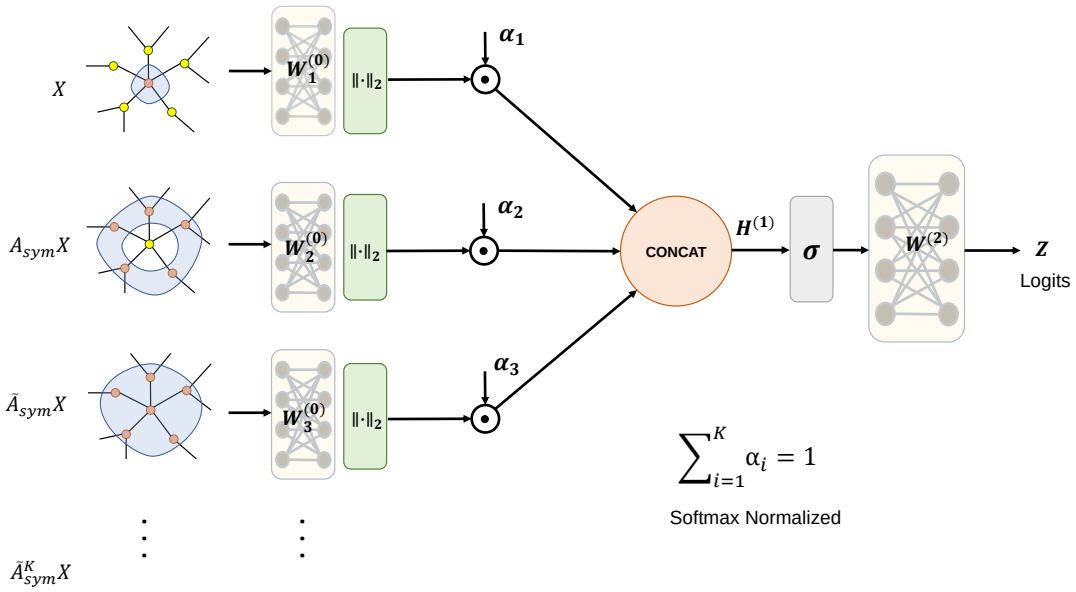


Figure 5.6: Figure shows model diagram of FSGNN. Input features are generated based on powers of A and $\tilde{A}$.

(FSGNN). Figure 5.6 shows the diagrammatic representation of our model. Input features are precomputed using A_{sym} and $\tilde{A}_{sym}$ and transformed using a linear layer unique to each feature matrix. L2-normalization is applied on the output activations of the first layer and weighted with scalar weights regularized by the softmax function. Output features are then concatenated and non-linearly transformed using ReLU and mapped to the second linear layer. Cross-entropy loss is calculated with output logits of the second layer. The model can be represented as,

$$Z = \sigma\left(\text{CONCAT}_l(\alpha_l X_l W_l^{(0)})\right) W^{(1)} \tag{5.7}$$

where $\text{CONCAT}_l, \forall\, l \in \{1 \,..\, L\}$, X_l are input features and σ is ReLU activation function.

5.5 Related Work

GNNs have emerged as an indispensable tool to learn graph-centric data. Many prediction tasks like node classification, link prediction, graph classification, etc. [Defferrard, Bresson, and Vandergheynst 2016][Kipf and Welling 2017] introduced a

simple end-to-end training framework using approximations of spectral graph convolutions. Since then, there has been efforts in the research community to improve the performance of GNNs, and a variety of techniques have been introduced. Earlier GNN frameworks utilized a fixed propagation scheme along all edges, which is not always scalable for larger graphs. GraphSAGE[Hamilton, Ying, and Leskovec 2017] and FastGCN[Chen, Ma, and Xiao 2018] introduce neighbor sampling approaches in graph neural networks. GAT [Velickovic et al. 2018b] introduces the use of the attention mechanism to provide weights to features that are aggregated from the neighbors. APPNP [Klicpera, Bojchevski, and Günnemann 2018], JK [Xu et al. 2018], Geom-GCN [Pei et al. 2020], SimP-GCN [Jin et al. 2021], and CPGNN [Zhu et al. 2021] aim to improve the feature propagation scheme within layers of the model. More recently, researchers are proposing to make GNN models deeper [Chen et al. 2020; Li et al. 2021; Godwin et al. 2021]. However, deeper models suffer from over-smoothing, where after stacking many GNN layers, features of the node become indistinguishable from each other, and there is a drop in the performance of the model. DropEdge [Rong et al. 2020] proposes to drop a certain number of edges to reduce the speed of convergence of over-smoothing and relieves the information loss. GCNII [Chen et al. 2020] use residual connections and identity mapping in GNN layers to enable deeper networks. RevGNN [Li et al. 2021] uses deep reversible architectures and [Godwin et al. 2021] uses noise regularisation to train deep GNN models.

Researchers find that traditional GNNs work well in homophily graphs but fail to generalize to heterophily graphs. Several models propose to handle heterophily properties in graph data: H2GCN [Zhu et al. 2020], CPGNN [Zhu et al. 2021], TDGNN [Wang and Derr 2021], Geom-GCN [Pei et al. 2020], and GPRGNN [Chien et al. 2021]. However, a recent work [Ma et al. 2021] reveals that GCNs can achieve strong performance on heterophily graphs under certain conditions.

Similar to our work, the idea of decoupling feature generation and representation learning has been adopted by several existing works, such as SGC [Wu et al. 2019a] and SIGN [Frasca et al. 2020]. However, these models are equivalent to `Single_feature` and `All_feature` setting, thus suffer from similar drawbacks. Many GNN models exhibit similarity of weighting the features, however, many of them have fixed weighting scheme like JK-Net [Xu et al. 2018], APPNP [Klicpera, Bojchevski, and Günnemann 2018], and GCNII [Chen et al. 2020]. In case of models with adaptable weighting scheme like ChebyNet [Defferrard, Bresson, and Vandergheynst 2016] and GPR-GNN [Chien et al. 2021], learned weights do not show feature selection pattern due to lack of explicit regularization. Moreover, these GNN models do not use no-loop features by design, thus limiting their learning capability on heterophily

Table 5.2: Statistics of the node classification datasets

Datasets	Hom. Ratio	Nodes	Edges	Features	Classes
Cora	0.81	2,708	5,429	1,433	7
Citeseer	0.74	3,327	4,732	3,703	6
Pubmed	0.80	19,717	44,338	500	3
Chameleon	0.23	2,277	36,101	2,325	4
Wisconsin	0.21	251	499	1,703	5
Texas	0.11	183	309	1,703	5
Cornell	0.30	183	295	1,703	5
Squirrel	0.22	5,201	198,353	2,089	5
Actor	0.22	7,600	26,659	932	5

datasets.

5.6 Experiments

In this section, we evaluate the empirical performance of our proposed model on real-world datasets on the node classification task and compare with other graph neural network models.

5.6.1 Datasets

For fully-supervised node classification tasks, we perform experiments on nine datasets commonly used in graph neural networks literature. Details of the datasets are presented in Table 5.2. Homophily ratio [Zhu et al. 2020] denotes the fraction of edges which connects two nodes of the same label. Cora, Citeseer, and Pubmed [Sen et al. 2008] are citation networks based datasets and in general, are considered as homophily datasets. Graphs in Wisconsin, Cornell, Texas [Pei et al. 2020] represent links between webpages, Actor [Tang et al. 2009] represent actor co-occurrence in Wikipedia pages, Chameleon and Squirrel [Rozemberczki, Allen, and Sarkar 2020] represent the web pages in Wikipedia discussing corresponding topics. These datasets are considered as heterophily datasets. To provide a fair comparison on experimental results, we use publicly available data splits taken from [Pei et al. 2020][1]. These splits have been frequently used by researchers for experiments in their publications. The results of comparison methods presented in this paper are also based on this split.

[1]https://github.com/graphdml-uiuc-jlu/geom-gcn

5.6.2 Preprocessing

We follow the same preprocessing steps used by [Pei et al. 2020] and [Chen et al. 2020]. Other models to which we compare our results also follow the same set of procedures. Initial node features are row-normalized. To account for both homophily and heterophily, we use the adjacency matrix and adjacency matrix with added self-loops for feature transformation. Both matrices are symmetrically normalized. For efficient computation, adjacency matrices are stored and used as sparse matrices.

5.6.3 Settings and Baselines

For a fully-supervised node classification task, each dataset is split evenly for each class into 48%, 32%, and 20% for training, validation, and testing [Pei et al. 2020; Zhu et al. 2020]. We report the performance as mean classification accuracy over 10 random splits.

We fix the embedding size to 64 and set the initial learnable scalar parameter with respect to each hop to 1. Thus, the initial scalar value α_i is set to $1/L$. Hyper-parameter settings of the model for best performance are found by performing a grid-search over a range of hyper-parameters. We train the model under two input settings. In first setting, we follow the conventional classification of the datasets as homophily datasets and heterophily datasets. For homophily datasets, we use input features as node's self feature and self-looped aggregated features. For heterophily datasets, we use self-features and no-loop aggregated features. In the second setting, we use all features to train the model.

We compare our model to 10 different baselines and use the published results as the best performance of these models. GCNII [Chen et al. 2020] and H2GCN [Zhu et al. 2020] have proposed multiple variants of their model. We have chosen the variant with the best performance on most datasets. GPRGNN uses random splits in their published results. For fair comparison, we ran their publicly available code on our standard splits while keeping other settings same. To get the best results, we performed hyperparameter search as mentioned in their code repository.

5.7 Results

5.7.1 Node Classification Results

Table 5.3 shows the comparison of the mean classification accuracy of our model and other popular GNN models. In general, traditional GNN models like GCN and

Table 5.3: Mean classification accuracy on fully-supervised node classification task. Results for GCN, GAT, GraphSAGE, Cheby+JK, MixHop and H2GCN-1 are taken from [Zhu et al. 2020]. For GEOM-GCN, GCNII and WRGAT results are taken from the respective article. Best performance for each dataset is marked as bold and second best performance is underlined for comparison.

		Cora	Citeseer	Pubmed	Chameleon	Wisconsin	Texas	Cornell	Squirrel	Actor	Mean Acc.
GCN		87.28±1.26	76.68±1.64	87.38±0.66	59.82±2.58	59.80±6.99	59.46±5.25	57.03±4.67	36.89±1.34	30.26±0.79	61.62
GAT		82.68±1.80	75.46±1.72	84.68±0.44	54.69±1.95	55.29±8.71	58.38±4.45	58.92±3.32	30.62±2.11	26.28±1.73	58.55
GraphSAGE		86.90±1.04	76.04±1.30	88.45±0.50	58.73±1.68	81.18±5.56	82.43±6.14	75.95±5.01	41.61±0.74	34.23±0.99	69.50
Cheby+JK		85.49±1.27	74.98±1.18	89.07±0.30	63.79±2.27	82.55±4.57	78.38±6.37	74.59±7.87	45.03±1.73	35.14±1.37	69.89
MixHop		87.61±0.85	76.26±1.33	85.31±0.61	60.50±2.53	75.88±4.90	77.84±7.73	73.51±6.34	43.80±1.48	32.22±2.34	68.10
GEOM-GCN		85.27	**77.99**	90.05	60.90	64.12	67.57	60.81	38.14	31.63	64.05
GCNII		88.01±1.33	77.13±1.38	**90.30±0.37**	62.48±2.74	81.57±4.98	77.84±5.64	76.49±4.37	N/A	N/A	-
H2GCN-1		86.92±1.37	77.07±1.64	89.40±0.34	57.11±1.58	86.67±4.69	84.86±6.77	82.16±4.80	36.42±1.89	35.86±1.03	70.71
WRGAT		88.20±2.26	76.81±1.89	88.52±0.92	65.24±0.87	86.98±3.78	83.62±5.50	81.62±3.90	48.85±0.78	**36.53±0.77**	72.93
GPRGNN		**88.49±0.95**	77.08±1.63	88.99±0.40	66.47±2.47	85.88±3.70	86.49±4.83	81.89±6.17	49.03±1.28	36.04±0.96	73.37
FSGNN (Homo/Hetero)	**3-hop**	87.61±1.39	77.17±1.48	89.70±0.44	78.93±1.03	88.24±3.40	**87.57±4.71**	87.30±5.93	73.86±1.81	35.38±0.81	78.42
	8-hop	88.23±1.17	77.35±1.17	89.78±0.38	**78.95±0.86**	87.65±3.51	**87.57±4.86**	87.30±4.53	73.94±2.02	35.62±0.87	**78.49**
FSGNN (All)	**3-hop**	87.73±1.36	77.19±1.35	89.73±0.39	78.14±1.25	**88.43±3.22**	87.30±5.55	87.03±5.77	73.48±2.13	35.67±0.69	78.30
	8-hop	87.93±1.00	77.40±1.93	89.75±0.39	78.27±1.28	87.84±3.37	87.30±5.28	**87.84±6.19**	**74.10±1.89**	35.75±0.96	78.46

GAT have higher performance on homophily datasets, however, they perform poorly on heterophily datasets. More recent models like H2GCN, WRGAT and GPRGNN perform relatively better on both homophily and heterophily datasets.

On heterophily datasets, our model shows significant improvements especially 51.1% on Squirrel and 18.8% on Chameleon dataset. Similarly, on Wisconsin, Texas, and Cornell, improvements are 1.6%, 1.2%, and 6.9%, respectively. On homophily datasets, we observe that different models perform best on different datasets. Our model still has consistent and comparable performance to SOTA.

5.7.2 Comparison with `Sub_feature`

In our work, we aim to maintain performance as close as possible to `Sub_feature` (hidden dimension, d = 64) in Table 5.1. On five datasets: Cora, Chameleon, Cornell, Squirrel and Actor, our model performs as well as `Sub_feature`, however for other datasets, performance is comparable, albeit a bit lower. During the training, our model starts with all input features and learns to identify relevant features and reduce the effect of irrelevant features. However, it is difficult to completely reduce the effect of noisy/irrelevant features without an explicit forgetting scheme. We consider the development of such a scheme to completely remove the impact of noisy features in GNN training as the future direction of our work.

5.7.3 Ablation Studies

In this section, we consider the effect of various proposed design strategies in section 5.3.1 on the performance of the model. In general, graph neural networks are sensitive to the hyperparameters used in training and require some amount of tuning to get the best performance. Since each dataset may have a different set of best hyperparameters, it can be difficult to judge design decisions based just on best performance of the model with single hyperparameter setting. To provide a comprehensive evaluation, we compare the average accuracy of the model over 1,080 combinations of the hyperparameters. The hyperparameters we tune are learning rate and weight decay of layers and dropout value applied as regularization between layers. Table 5.4 shows the average of classification accuracy values under various settings.

For most datasets, our proposed design schemes lead to better average accuracy. Cora and Citeseer show better average performance without softmax regularization; however, the peak performance is marginally less with regularization. Even though Wisconsin shows higher average accuracy without normalization, however, the best performance on the dataset was achieved with the normalization layer. We found

Table 5.4: Ablation study over 1080 different hyperparameter settings.

	Cora	Citeseer	Pubmed	Chameleon	Wisconsin	Texas	Cornell	Squirrel	Actor
Proposed	83.68±2.22	74.48±1.44	**89.24±0.27**	**72.48±4.16**	81.48±5.62	**78.80±5.88**	**78.09±2.22**	**63.57±6.83**	33.54±1.21
Without soft-selection	**87.07±0.26**	**76.45±0.27**	89.09±0.39	72.27±1.34	78.03±6.55	76.28±6.72	74.32±6.54	61.73±4.15	34.15±0.64
Common weight ($W^{(0)}$)	83.19±1.41	72.15±1.02	88.96±0.28	68.24±6.03	70.56±10.94	68.45±7.65	68.18±9.13	56.63±8.54	32.73±1.48
Without L2-normalization	77.12±3.49	71.40±10.01	87.72±0.77	53.06±6.18	**82.60±2.68**	76.33±3.87	76.18±3.43	32.60±6.38	**36.66±0.55**

that Actor was the only dataset where accuracy was reduced with the addition of the normalization layer. Without the normalization layer, our model achieves 37.63% accuracy. However, to maintain consistency, we do not include it in the main results. These variations also highlight that a single set of design choices may not apply to all datasets/tasks and some level of exploration is required.

It is interesting to note that performance on almost all datasets is sensitive to the choice of the hyperparameters for training the model as there is a wide gap between best and average performance. One exception is Pubmed, where the model's performance is relatively unperturbed under various hyperparameter combinations.

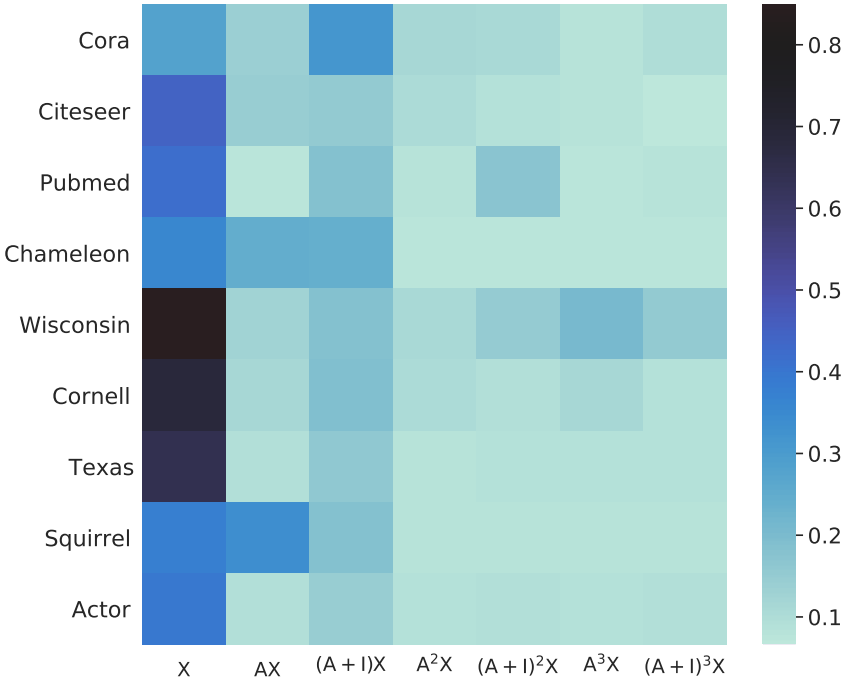


Figure 5.7: Heatmap of average of learned soft-selection scalar for all datasets

5.7.4 Soft-Selection Parameter Analysis

We analyze the learned soft-selection parameters on average over different model hyperparameter combinations. We use four different settings: 1) Proposed model setting, 2) without softmax regularization on scalar weight parameters, 3) shared linear transformation layer on input features, and 4) without L2-normalization on input

feature activations. For homophily datasets, it is easy to see that self-looped features are given more importance. Among heterophily datasets, Wisconsin, Cornell, Texas, and Actor have the most weights on node's self features. In these datasets, graph structure plays a limited role in the performance accuracy of the model. For Chameleon and Squirrel datasets, we observed that the node's own features and first-hop features (without self-loop) were more useful for classification than any other features.

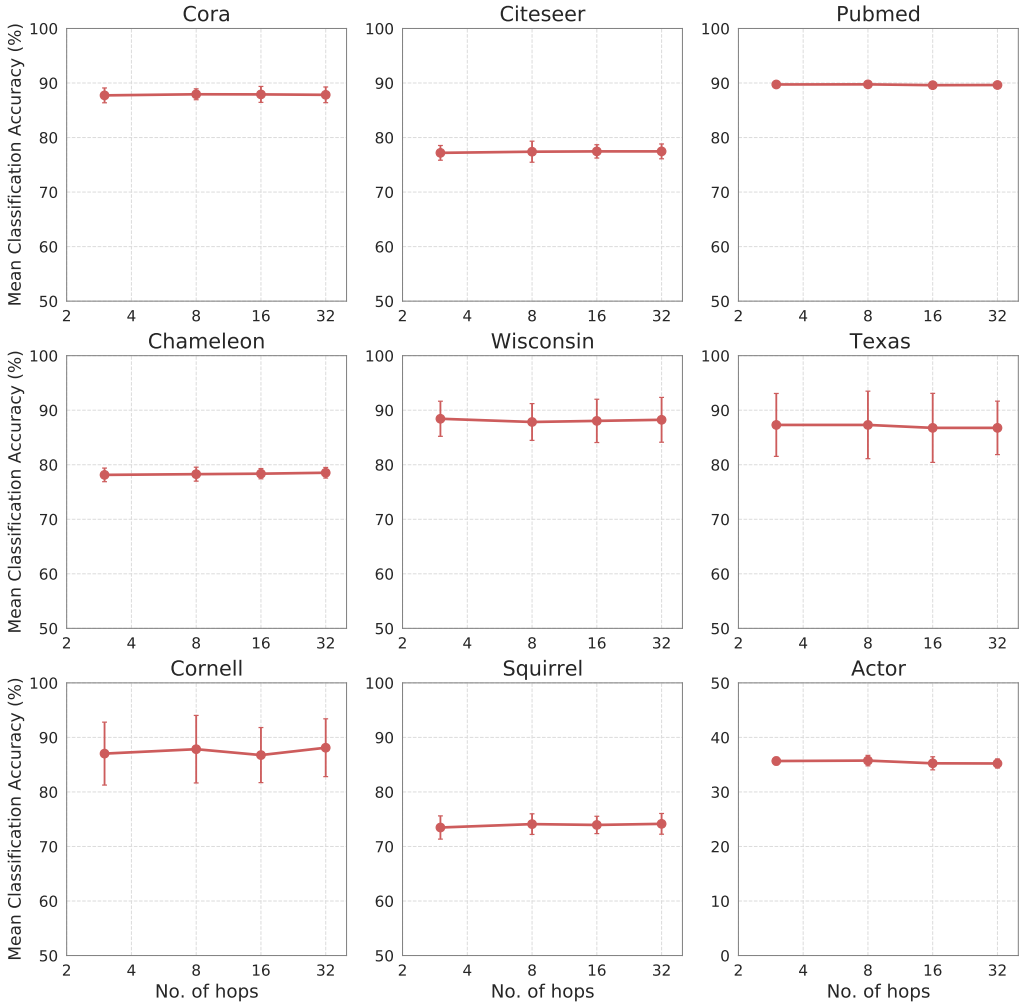


Figure 5.8: Figure shows the effect on classification accuracy of FSGNN with increase in the number of hops of feature aggregation. x-axis is in logarithmic scale.

5.7.5 Over-smoothing Analysis

Many GNN models suffer from the over-smoothing problem when the number of hops for feature aggregation is increased. In section 5.3.2, we discussed how feature selection can be helpful to overcome over-smoothing problem. In this section, we evaluate the change in model's performance with increase in the hops for aggregation. We run additional experiments with hop values set to 16, and 32 with all features as input as described in Section 6. Figure 5.8 shows the performance of the model for hop setting of 3,8,16 and 32. We observe that there is little variation in the performance of the model on various datasets and the model does not suffer from over-smoothing. This result is intuitive as aggregated features from higher hops are not very useful, and the model can learn to place low weights on them. For few datasets, we were able to achieve higher accuracy values: Citeseer (77.46%), Cornell (88.11%) and Squirrel (74.15%).

5.8 Conclusion

In this work, we explore the importance of feature selection in GNN training. We run extensive experiments to investigate GNN model design requirements for homophily and heterophily datasets and how feature selection can lead to higher prediction accuracy on benchmark datasets. Our experimental observations provide a definite confirmation that feature selection is a good direction for the exploration of GNN architectures. Based on our experimental observations, we propose a novel GNN model called FSGNN. Using extensive experiments, we show that simple 2-layered FSGNN outperforms the current state-of-the-art GNN models with complex designs on the node classification task. Analysis of the learned parameters provides us the crucial information of feature importance. In addition, we show that even shallow models can learn and provide high prediction accuracy, and with our model over-smoothing phenomenon can be easily avoided.

Chapter 6

Discussion

In this chapter, we cover several additional points related to the proposed GNN models on node ranking and node classification. We discuss the design decisions of these GNN models, their implications, and limitations. We provide details on possible applications of our models on real-world tasks. Finally, we combine our models under a unified framework to show the usefulness of domain-specific aggregation schemes in graph neural networks.

6.1 GNN model for Ranking

6.1.1 Design Considerations of Ranking Models

In Chapter 4, we explore the problem of approximating node ranking measures Betweenness Centrality and Closeness Centrality. Both ranking measures are based on the calculation of pairwise shortest paths between nodes in the given graph. This calculation is computationally expensive and requires significant execution time. To reduce the computational time for node ranking approximation, we propose to train GNN models to learn to approximate these ranking measures and inference on new graphs in significantly less time. Our model includes three unique design considerations as follows:

I. Finding nodes lying on possible shortest paths

As a pre-processing step, we identify nodes that have no shortest paths going through them. As these nodes do not contribute to the calculation of ranking measures, we exclude their effect on GNN aggregation operation by modifying the adjacency

matrix. This step is essential as it leads to the flow of feature information in the GNN model along possible shortest paths in the graph. Here, we actively avoid unnecessary aggregation of feature information which might be detrimental to the learning of the GNN model for node ranking.

II. Using simple Adjacency Matrix

In most GNN models, self-looped version of the adjacency matrix is used by adding a self-loop to all nodes in the graph. This process leads to cumulative aggregation of node features (discussed in Chapter 4), and is useful in tasks like node classification [Kipf and Welling 2017], but not useful in node ranking problem. For our model, we want to avoid cumulative aggregation of features and would like to map features of nodes lying at different hops separately. Using a simple adjacency matrix in the GNN model helps to achieve this result.

Table 6.1: Table shows comparison of ranking performance with KT Score values between proposed model and modified model with self-loops to non-zero centrality nodes.

	ER		**SF**		**GRP**	
	Original	**With self-loops**	**Original**	**With self-loops**	**Original**	**With self-loops**
Betweenness	0.902±0.03	0.596±0.16	0.976±0.01	0.960±0.01	0.899±0.04	0.513±0.08
Closeness	0.907±0.03	0.814±0.06	0.903±0.01	0.877±0.02	0.979± 0.01	0.880±0.01

Table 6.1 shows the comparison of our proposed GNN models with self-looped and simple adjacency matrix for approximation of Betweenness and Closeness centrality on synthetic graph datasets. KT Score for the GNN model with self-looped adjacency matrix is significantly lower than the one with just the simple adjacency matrix. The results indicate that cumulative aggregation is indeed not suitable for the GNN model and separating node features from different hops helps to learn node rankings better.

III. Separating Incoming and Outgoing Paths to the Nodes

Since we are accumulating neighborhood information for each node in a directed graph with incoming and outgoing edges, we separate the feature aggregation process into two parts. Neighborhood aggregation for neighbors on incoming paths and outgoing paths is done separately, and relevant scores are combined at the final stage of the model's predictions.

6.1.2 Implications of the model design

Default aggregation scheme in GNN models aggregates feature information flows along edges of the graph similar to the breath first search (BFS) algorithm. Our model differs from the standard GNN model with three different design considerations. With these design modifications, we explicitly introduce inductive bias of calculation of centrality measures that is not possible with the default aggregation scheme of GNN models. With design modifications I and II, we also exclude node features that can act as noise for the learning task of node ranking. Hence, our proposed models `GNN-Bet` and `GNN-Close` use a selective aggregation scheme that aims to approximate information flow along possible shortest paths in the graph.

Another design consideration of the models pertains to the loss function during the training of the model. Our model uses only ranking loss, more specifically `TripletMarginLoss` from `PyTorch` to learn to rank the nodes. Another possible way to approach the learning problem is to use error losses like `Mean Absolute Error (MAE)` or `Mean Square Error (MSE)` with respect to the real centrality values of the nodes. However, this approach has a challenging problem. In many cases, the absolute values of centrality measures in a graph can vary by large factors. For example, in Table 6.2, we have shown the maximum and minimum values of normalized centrality values for different types of synthetic datasets. We observe that the maximum value is many order times higher than the minimum value. The range of variation can go higher with larger graphs. Such a large variation makes it challenging to train a neural network model on MAE or MSE loss to approximate the absolute centrality values.

Table 6.2: Maximum and minimum values of betweenness centrality and closeness centrality for synthetic datasets.

	Betweenness Centrality		**Closeness Centrality**	
	Max Value	**Min Value**	**Max Value**	**Min Value**
ER	7.37×10^{-3}	1.28×10^{-10}	0.183	1.07×10^{-5}
SF	2.66×10^{-2}	1.89×10^{-13}	0.316	1.00×10^{-5}
GRP	9.01×10^{-5}	1.09×10^{-10}	0.060	1.04×10^{-10}

To experimentally verify our observations, we train our models with Mean Absolute Error(MAE) and Mean Square Error(MSE) loss functions and report the results in Table 6.3. We observed lower ranking performance with respect to KT Score. Even though the magnitude difference $\Delta S = MEAN(|\ S_{predict} - S_{true}\ |)$ is quite small, the wider range of centrality values still makes it difficult to achieve high

performance. We also experimented with a combination of ranking loss along with the MAE/MSE loss function. However, we still observed a lower performance of the model. Therefore, for our model, we exclusively use ranking loss during the training process.

Table 6.3: Table shows the KT score and ΔS for the model GNN-Bet and GNN-Close with MAE and MSE loss functions for synthetic datasets.

		GNN-Bet		**GNN-Close**	
		KT Score	**ΔS**	**KT Score**	**ΔS**
MAE Loss	**ER**	0.587 ± 0.134	$6.0 \times 10^{-5} \pm 5.2 \times 10^{-5}$	0.616 ± 0.228	$9.1 \times 10^{-3} \pm 2.9 \times 10^{-3}$
	SF	0.589 ± 0.073	$1.6 \times 10^{-5} \pm 7.9 \times 10^{-6}$	0.667 ± 0.054	$2.7 \times 10^{-3} \pm 1.9 \times 10^{-3}$
	GRP	0.479 ± 0.045	$1.0 \times 10^{-6} \pm 7.8 \times 10^{-7}$	0.579 ± 0.108	$8.0 \times 10^{-4} \pm 5.0 \times 10^{-4}$
MSE Loss	**ER**	0.246 ± 0.351	$7.6 \times 10^{-5} \pm 4.6 \times 10^{-5}$	0.662 ± 0.082	$8.3 \times 10^{-3} \pm 3.9 \times 10^{-3}$
	SF	0.291 ± 0.070	$3.4 \times 10^{-5} \pm 1.9 \times 10^{-5}$	0.602 ± 0.064	$2.9 \times 10^{-3} \pm 1.5 \times 10^{-3}$
	GRP	0.379 ± 0.049	$2.6 \times 10^{-6} \pm 6.0 \times 10^{-7}$	0.635 ± 0.062	$1.4 \times 10^{-3} \pm 5.0 \times 10^{-4}$

6.1.3 Limitations of the models

In the results section of Chapter 4, we demonstrate that our proposed GNN models significantly outperform traditional algorithms in node ranking while taking less execution time. However, there are still some limitations to our models. In this section, we provide details on some of these limitations:

Applicable to limited node ranking measures

Our proposed model is based on approximating shortest paths in a graph using graph neural networks. This aspect of the model is useful for approximating centrality measures like betweenness and closeness centrality. However, there are many other centrality measures like PageRank, Katz centrality, Eigenvector centrality, Degree centrality etc. that do not follow the assumption of information flow along the shortest paths in the graph. Hence, our modified aggregation scheme is not suitable for all node ranking measures and would require further modifications to make it suitable for other ranking measures.

Prediction of centrality values

In our proposed models, we aim to approximate the ranking of nodes and compare them with the ranking of real centrality values using Kendall's Tau score as a ranking measure. While the output values of the model are useful for predicting the ranking

of the nodes, they do not match with the actual betweenness centrality or closeness centrality values. As we already discussed in Section 6.1.2, high variability in real centrality values makes their prediction by the GNN model a challenging task. However, for many tasks, absolute value of centrality may have limited importance, and the relative ranking of the nodes might provide more useful information. Therefore, the utility of the model might vary with respect to the task at hand.

Prediction of top-k nodes

Determining the centrality value of a node is a purely numeric calculation process. Some recent works [Bisenius et al. 2018; Bergamini et al. 2019; Borassi and Natale 2019] have taken advantage of approximation algorithms to compute top-k centrality values with certain guarantees. For GNN models or any other neural network models in general, the quality of predictions is often tested on a test set, and no guarantees can be provided for the performance of the model on any input data. With our experiments, we verified that our model can provide good top-k performance when the training dataset is similar to the test dataset in terms of graph structural properties. Otherwise, the top-k performance of the model may suffer due to sub-optimal learning.

6.1.4 Applications

Betweenness and Closeness centrality measures are useful to identify nodes that play an important role in the spread of information across the graph. In this section, we discuss applications and scenarios where our proposed ranking model can be utilized.

Small graphs vs large graph datasets

We discussed in Chapter 4 that shortest-path measures like betweenness centrality and closeness centrality are computationally expensive, and therefore, there are multiple works in the literature to approximate these measures. However, when tackling real-world problems, we should determine the size of the graphs to select the best way to calculate node centrality values. For graphs up to hundreds to a few thousand nodes, modern computing hardware is powerful enough to compute these measures in a short amount of time. However, for larger graphs, we can select approximation algorithms, including our proposed model.

Influential nodes in Social Networks

One important field our ranking model can find its application is the analysis of social networks. Social networks consist of large graphs with often hundreds of thousands to millions of nodes. In addition to being large, they are also dynamic by nature i.e., graph structure changes rapidly, and the nature of information flow might change along with it. Social networks have become an integral part of our society and have the powerful capability to reach a massive number of users within a short period of time. They also have significant power to spread information and affect people's political and non-political opinions. However, not all users play a significant role in the network, and identifying influential users can help to spread useful information (e.g., Ad networks) and reduce the spread of misinformation (e.g., fake news). With our proposed model, we can quickly identify such influential nodes.

Another challenge is the rapid change in the graph structure. In Chapter 4, we also propose an application of our model in dynamic networks. Due to short inference time, our model can predict influential nodes in rapidly changing networks.

Potential applications and future work of our model

One major disadvantage of centrality measures is that they rely purely on the graph structure to identify influential nodes. Most graph-structured data also includes node/edge attribute information that can provide more context toward ranking the importance of the nodes. However, these measures fail to utilize these attributes. Our proposed models `GNN-Bet` and `GNN-Close` by default initialize random unique embeddings for the nodes and learn the ranking of the nodes based solely on the graph structure. Similar to other GNN models, these models can also include node and edge attributes as additional features and learn to rank the nodes in a broader context. We have not explored it in our work and consider it as a future direction where we can expand the utility of our model.

6.2 GNN model for Node Classification

In this section, we review design choices for our proposed model `FSGNN` for the node classification task and discuss the implications of our choices on the model's performance. We then discuss the limitations of our model and provide details on possible applications of the model on real-world tasks.

6.2.1 Design considerations of the model

The architecture of `FSGNN` model is based on many different considerations while aiming to improve prediction accuracy and scalability over large datasets. Some key design highlights of the model are as follows.

Homophily and Heterophily in graph datasets

Earlier GNN models like GCN [Kipf and Welling 2017] were based on the assumption of homophily as discussed in Chapter 5. However, recent papers in literature have proposed more complex and varied heterophily datasets [Pei et al. 2020; Lim et al. 2021] to benchmark GNN models on node classification task. Availability of these new datasets has been challenging for many GNN models as they fail to perform well and often show poor prediction accuracy [Lim et al. 2021]. The main reason behind the poor performance is strict assumptions of properties of datasets (in this case, homophily) that cause them to be less robust. Real-world datasets can often have varying levels of homophily/heterophily properties; hence the optimal GNN model needs to be versatile enough to learn on these properties. Our proposed model considers the possibility of both homophily and heterophily in datasets and learns to select best features corresponding to the node labels.

Pre-computation of node features

Most GNN models combine feature generation and feature learning operations within the GNN layer. Some GNN models [Wu et al. 2019a; Frasca et al. 2020] have taken a different approach of pre-calculating node features before using neural network to learn on the data. The motivation for these models to pre-calculate node features is to either study the effectiveness of learnable weights at each GNN layer or to improve the scalability of the GNN model. In our case, we also use the same approach; however, the motivation is to generate node features that are suitable for both homophily and heterophily characteristics in the data. We use both self-looped and simple adjacency matrices to achieve our goal. The ability to scale model for larger datasets provides an additional benefit of pre-calculating node features.

Learning to identify important hop-features

In our proposed model, we pre-compute node features by feature aggregation over multiple hops. The number of hops is selected as a hyper-parameter for the model. In our experiments in Chapter 5, we set this hyper-parameter as 3 and 8 for all datasets. After the feature aggregation step, we get a set of features for each node.

However, all features are not useful, and some of them might be unrelated/noisy with respect to the node labels. Hence, we propose to learn to select only features that are optimal for higher performance of the model. With our experiment results, our proposed models in both Chapter 5 and Appendix C show that learning to select the best node features while discarding noisy features helps to improve the performance of the GNN model.

6.2.2 Implications of the model design

In Chapter 5, in Table 5.1 we show empirical results that with default aggregation scheme in GNNs, we do not always get optimal results, and some type of feature selection scheme is required to avoid learning on noisy data. With our specific design considerations of `FSGNN`, we are able to train the model to learn specifically on useful features for the node classification task. Instead of using the default aggregation scheme in GNN, we are using adaptive hop selection of features that leads to higher accuracies, and our model outperforms other state-of-the-art GNN models.

With keeping fundamental design considerations in mind, we propose `Dual-Net GNN` model, which is different from `FSGNN` both in terms of model architecture and training procedure. However, we still keep the principle of choosing the best hop features. Once again, our model outperforms other GNN models. With the experiment results of both models, we empirically confirm that a learnable selective feature aggregation model is an improvement over the default aggregation model for the node classification task.

6.2.3 Limitations of the proposed model

Our proposed models `FSGNN` and `Dual-Net GNN` outperform other SOTA GNN models on many node classification datasets. However, these models still suffer from some limitations which are described as follows.

Unavailability of graph regularization measures

In recent years, in addition to novel GNN architectures, many training regularization techniques have been proposed in the literature. Some of these techniques are DropEdge [Rong et al. 2020], which randomly drops edges of the input graph in each forwards propagation, DropNode [Feng et al. 2020; Papp et al. 2021], which drops nodes of the input graph randomly and independently, embedding smoothing with random propagation [Feng et al. 2020] etc. These techniques are similar to Dropout [Srivastava et al. 2014] regularization but on graph structure and have been

experimentally shown to improve the performance of the GNN models. Since in our models, we pre-calculate node features with the fixed graph structure, our models cannot utilize these regularization techniques to improve performance further.

Memory requirements for pre-calculation of features

One advantage of our proposed model is the ability to train on large graph datasets with small input batches on GPU while utilizing complete graph information. Hence, the training of our model is significantly faster compared to other GNN models, which cannot be trained on GPU with full graph structure or would need to randomly sample nodes in each iteration of training. However, the initial step of pre-calculation of node features required multiplication of the adjacency matrix with the node feature matrix. For large graph datasets, a sparse adjacency matrix can occupy tens of GBs, and a node feature matrix can occupy 100GB+ memory. Therefore, in such cases, for the pre-calculation of node features, higher CPU memory is required. However, this requirement arises only once, and later smaller GPU memory is sufficient for training the model.

6.2.4 Applications

Our proposed models have been specifically designed and trained for the node classification task. Though our improved aggregation mechanism might be applicable for other graph learning tasks like link prediction, graph classification, graph clustering, etc., we restrict the discussion of applications to node classification.

Node classification task, in general, is predicting unknown labels of the node based on available node/edge feature data and the graph structure. These node labels can often be attributes of the users or properties of entities that are represented as nodes. Though the problem of node classification can be defined in a wide variety of fields, we limit the discussion of applications to tasks that are often used by the community as a benchmark for GNN models.

Prediction of user attributes in Social Networks

In social networks, users connected to each other via friendship often share common attributes. These attributes can be geographical location, workplace, political affiliations, etc. These attributes can be considered as labels of the nodes and can be predicted for users who have not provided this information in their profile. With this information, social media companies can introduce new people to users who share common interests, thus effectively increasing their social circles.

Predictions of User Interests in Recommender Systems

One of the biggest challenges of e-commerce companies is to suggest products to users that are relevant to their interests. For example, a tech-enthusiastic user might be interested in tech products like CPUs, GPUs, monitors, VR headsets, etc. By building profiles of users and connecting users with similar purchase histories, new users can be classified into categories relevant to their interests. This simplifies the challenge of suggesting new products out of hundreds of thousands of products available on the websites and making the process of purchasing a product easier for both seller and buyer.

Labeling papers in Citation Networks

One of the largest benchmarking node classification datasets [Hu et al. 2021] tries to solve the problem of classifying research papers. There are millions of papers published every year, and classifying them into research field categories is a challenging problem. GNN models solve this problem as a node classification task. As research papers often cite multiple papers of the same field, GNN models can learn to classify labels of papers by aggregating features of nodes via edges in the citation network. Nodes sharing similar features will have the same labels and vice versa. In Appendix B.2, we show that our model outperforms other GNN models on the task of predicting labels of nodes in a large citation network.

6.3 Unified Framework for Graph Neural Networks

In this section, we review our proposed models and analyze them from a common unified perspective. We then discuss the necessity of task-specific adaptive aggregation schemes and their role in improving the performance of graph neural networks.

6.3.1 Commonalities between our proposed GNN frameworks

In Sections 6.1 and 6.2, we discussed specific strategies for designing architecture of our proposed GNN models. The strategies and the tasks they were applied to are significantly different from each other. However, they share a common approach to improving the model's performance by modifying the default aggregation scheme of graph neural networks.

`GNN-Bet` and `GNN-Close` use feature aggregation via possible shortest paths and `FSGNN` and `Dual-Net GNN` models use learnable hop feature selection as their fea-

ture aggregation scheme. The common strategy of all models is to avoid unnecessary/noisy node features and rely on useful high-signal features to learn on a given task. This approach is similar to feature selection methods [Tang, Alelyani, and Liu 2014; Li et al. 2017a] where a subset of features is selected from given input data that can maximize the signal-to-noise ratio present in the data.

As we have already shown in our experiments that completely relying on neural network models to learn on noisy data may not be optimal. The use of explicit schemes that can help to filter out noise are significantly more helpful and leads to better-performing model. Considering that the use of graph neural networks has been spreading into a wide variety of fields, always using the default aggregation scheme might restrict the utility of GNNs. Similar to the example of homophily and heterophily in social networks, different fields have different characteristics present in the data. Since feature aggregation is a fundamental operation of GNNs, by leveraging domain-specific knowledge, we can enhance the capabilities of GNNs by integrating this knowledge into the architecture design of novel GNN models.

6.3.2 Challenges of graph-structured data

Proposing domain-specific aggregation scheme for graph neural networks paint a pessimistic picture of graph learning. As the deep learning community is trying to develop general-purpose models that can be applied to any area, using domain-specific GNN models seems like a step back.

However, we argue that graph learning problems have their own unique challenges when compared with other learning tasks like vision or natural language processing (NLP). For example, for learning visual tasks, the same type of objects will have similar characteristics irrespective of the source of the image. For example, in Figure 6.1, the vision model can be trained on data from different sources but can successfully detect entities like humans, planes, buses, animals etc. [Lee and Park 2020]. Recently large image models have been successfully trained on huge datasets, and features extracted from these models can be used for a wide variety of vision tasks.

Similarly, text data also contains similarly consistency irrespective of the source. Word meanings, grammar rules, word positions, and contexts of words with respect to each other remain consistent enough to enable language models to learn and perform various tasks. Accurate machine translation of the 100 languages (Figure 6.2) is a great example of powerful and generalizable NLP models [Fan et al. 2021]. A huge corpus of data with billions of sentences can make the model generalizable for various NLP related tasks.

In the case of graphs, graph structure alone reveals limited information, and the

Figure 6.1: Instance segmentation task with vision models accurately detects objects of various categories in the images. Source [Lee and Park 2020].

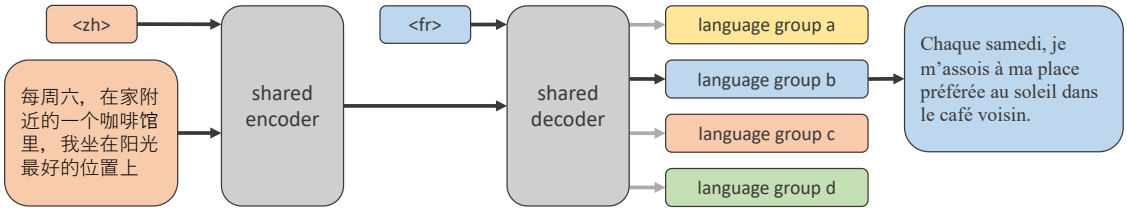


Figure 6.2: Language translation model proposed by [Fan et al. 2021] can translate any pair of 100 languages. Researchers used corpus of billions of sentences to train the model.

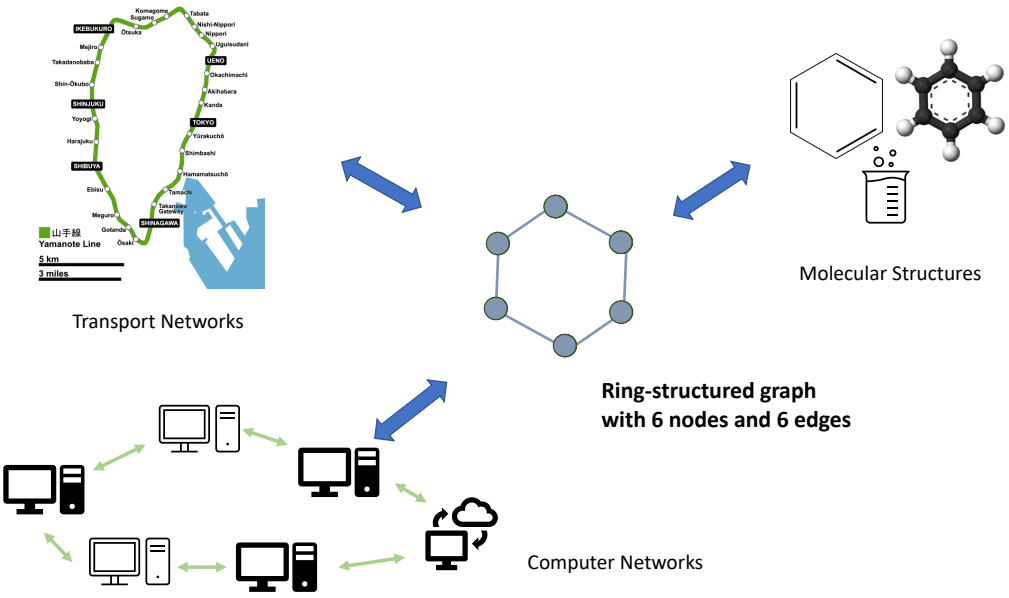


Figure 6.3: Data from different origins can be used to construct same graph. In this case, train network of Yamanote Line (Tokyo) with 6 major stations, chemical structure of Benzene, and computer network with 6 computation terminals form a ring-shaped graph. The structure of graph remains same, however, node and edge attributes will differ significantly.

context of the dataset or application is required for successful learning. Figure 6.3 shows data sources of different origins can lead to the same ring-structured graph. While the ring structure does encode some information, however, the context of source of data whether it is a chemical structure or a transport network brings significantly different conclusions. Similar observations can be derived from different types of motifs in the graph that can be generated from the data.

6.3.3 General vs domain-specific approach

In Section 6.3.2, we described the challenges associated with graph-structured data and the need to have the context of task and source of data for successful graph learning. While pre-training and transfer learning has seen tremendous success in vision and NLP models, such successful implementations on graph-structured data have yet to be seen. There have been domain-specific pre-training approaches [Hu et al. 2019], but it is a challenging task to carry their benefits to other domains.

In addition to the challenges arising in simple graphs, graphs themselves can be categorized into many types, like directed graphs, undirected graphs, heterogeneous graphs, multi-layered graphs, hyper-graphs, dynamic graphs, temporal graphs, etc. We already have numerous papers in literature proposing variants of graph neural networks specialized for different types of graphs. Often these variants attempt to leverage the unique characteristics in the structure of these graphs.

Above mentioned challenges and approaches make the case to approach the graph learning problems in a domain-specific way. We can introduce novel graph aggregation schemes relevant to the task and design the GNN architecture to leverage domain-specific knowledge for better predictions.

6.3.4 Possible approaches for designing feature aggregation schemes

So far, we have established the need for adaptive feature aggregation schemes for domain-specific tasks. This leads to the question of how to approach the design of a new aggregation scheme. There can be many different approaches based on the field, and we explore some of them as follows.

Information flow based aggregation

Feature aggregation schemes can be designed based on the assumptions of information flow in the graph. For our proposed node ranking models, we followed the assumption of information flowing through the shortest available paths. Similarly, for other types of networks, the information may transmit in a different way. Examples of such networks are heterogeneous information networks and multi-layered networks. These networks with different types of nodes, edges, and connectivity patterns might have information flow different than a simple graph. Hence, based on the application relevant aggregation scheme can be designed.

Data characteristics based aggregation schemes

The characteristics of nodes and edges play a significant role in the feature aggregation step. The presence of homophily and heterophily in neighboring nodes guided the design of our models `FSGNN` and `Dual-Net`. Study of these characteristics can help us decide which features are useful and which of them are noisy, and an adaptive aggregation scheme can be designed. Another example is in e-commerce systems, where the graphs can exist within products, users, and in-between them. A user can have multiple interests, and a product can belong to multiple categories; therefore,

a selective aggregation scheme can provide a better recommendation to the user and improve revenue for the companies.

Sub-graph based aggregation schemes

So far, we have discussed the aggregation of features within individual nodes. However, in some applications, feature aggregation between groups of nodes or a sub-graph can also be considered. One example is mapping chemical structures to graphs in drug-discovery applications. Atoms are represented as nodes, and chemical bonds between them as edges in the graph. In these chemical structures, functional groups like alkyl, benzyl, hydroxyl, etc., are a collection of atoms that have unique properties. Hence, in the graph, these functional groups can be denoted as sub-graphs, and the feature aggregation step can be modified between these sub-graphs and other nodes. A similar approach can be taken in other applications where a sub-graph can play a unique role in the graph.

6.4 Relationships between Graph Learning Tasks

Graph learning encompasses many different types of tasks: node classification, link prediction, node ranking, graph classification, graph regression, and so on. While these tasks may seem different from each other, the learning process on these tasks often shares similar principles. For example, the link prediction task requires identification of nodes with similar attributes that are likely to be connected to each other. Likewise, the task of node classification needs to establish similarity between nodes so that the labels of nodes can be predicted.

Graph neural networks have been applied to many different tasks, and the general implementation requires features aggregation or message passing step. Using the same basic principles, GNN models generate node embeddings or graph embeddings that can be utilized in different downstream tasks.

6.4.1 Relationship between Node Ranking and Node Classification

In our thesis, we focus on the tasks of node ranking and node classification. The first task emphasizes that some nodes play a more important role than other nodes in the graph. The second task emphasizes the role of connectivity between nodes to demonstrate the similarity or dissimilarity of the nodes.

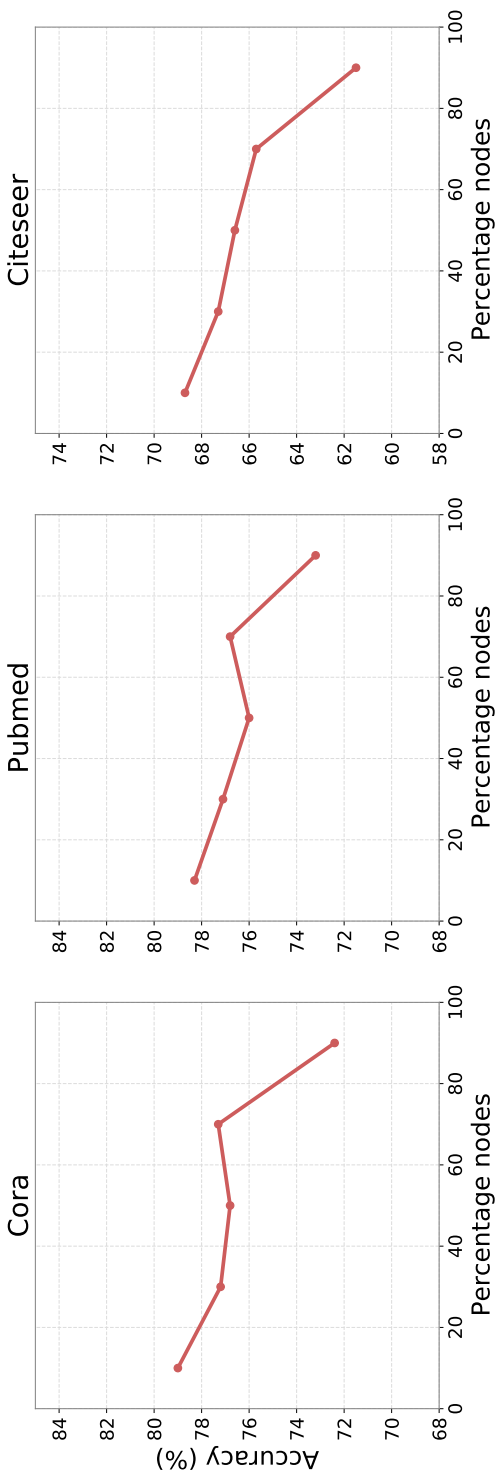


Figure 6.4: Nodes are sorted with respect to the betweenness centrality values and binned in 20% increments. GNN model is trained with nodes in each bin disconnected from the graph and model's performance is measured.

Node ranking measures rank the nodes based on the structure of the graph. High-ranking nodes have a higher capacity to spread information or act as an information bridge between clusters of nodes. Using this concept in feature aggregation step of GNNs, high-ranking nodes also contribute to larger extent to pass feature information to other nodes. Hence, such nodes can have a significant effect on the performance of the GNN model. To verify this effect, we conducted experiments on three commonly used node classification datasets: Cora, Citeseer and Pubmed. The nodes are sorted with respect to betweenness centrality values, and step-by-step, we disconnect 20% of sorted nodes at a time. We choose a larger percentage value for two reasons:

(1) We train the model with 3-hop aggregation as a standard practice reported in the GNN literature. As the graph sizes are relatively smaller, aggregating features over 3 hops sufficiently diffuse feature information in the graph. The effect of disconnecting a smaller number of nodes or a single node might get hidden due to compensation by other nodes. With a relatively larger number of nodes disconnected, we can compare the effect of node ranking in a more effective way.

(2) Choosing a larger number of nodes (i.e., 20%), we can observe more noticeable variations in performance values and confirm that the phenomenon arises due to the relationship between node ranking and classification and not due to fluctuations that may arise due to the hyper-parameters of the model.

After diconnecting the nodes, we train the GNN model on the resultant graph and measure the node classification performance. Figure 6.4 shows the performance of the GNN model under various settings. The x-axis denotes the percentage of nodes in sorted order, for example, 0-20% range denotes 20 percent nodes with the lowest betweenness centrality values and so on. We observe that as we remove nodes with subsequent higher centrality values, the accuracy of the GNN model decreases on both datasets.

Such insights can be useful in cases where graph structure is known, and there is limited availability of feature information of nodes and can be queried with a limited budget. In the real-world, such scenarios exist in the form of multi-layer networks, where there are multiple graph networks for the same nodes but from different domains. For example, user connectivity over the social network is known, but the availability of financial information of users is limited. The above method can then be used to select specific users to query financial information and infer that information (as labels) for other users.

However, the relation between node ranking and the model's performance on node classification may not always be correlated on real-world data. For many real-world

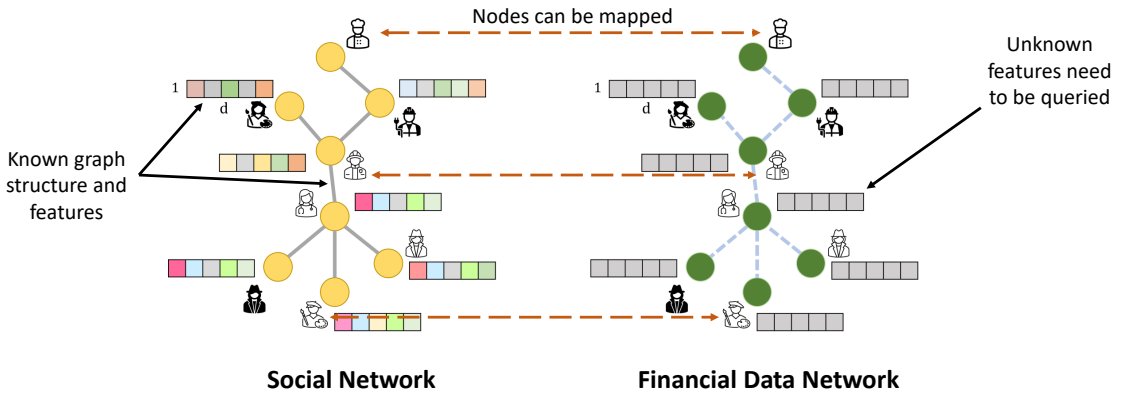


Figure 6.5: Multi-layer networks can often share similar graph structure and can be used to infer missing attributes of the nodes.

datasets, features of the nodes play a more important role than the graph structure for classification performance. In such cases, message passing by high-ranking nodes may not play a significant role, and utilizing ranking information may not be useful. Hence, when learning on any real-world data, it is essential to verify how much graph structure can play a role in the model's performance.

6.4.2 Application utilizing node ranking and classification

In literature, most models often focus on a single task. However, in the real world, data analysis often requires the use of multiple approaches/techniques. One such application is the prediction of user data in multi-layer networks. Multi-layer or multiplex networks have the same user as a node on multiple networks. For example, the same user can join two social networks, Facebook and Twitter, and exhibit similar connectivity patterns with friends on both social networks. Similarly, users can be part of multiple networks like social networks, financial data networks, communication/mobile networks and, so on. Figure 6.5 shows an example of a multi-layer network consisting of a social network and a hidden financial data network. Users who are friends in the social network can also share geographical locations or other social attributes. Such users can also share similar financial information. While social network data is easily available or queried, querying financial data can be expensive or prohibitive on large scale. By mapping the social network, we can get information on the graph structure, user's attributes, and influential or well-connected users in the community. Using this data, a shadow network can be created for financial data and only data of selected influential users can be queried. By learning on this data,

unknown attributes of other users can be predicted. Such an approach can be used for targeted social welfare programs that are often already resource-constrained and make it easy to provide benefits to the individuals.

Chapter 7

Conclusion

Graph Neural Networks are becoming one of the important tools in deep learning similar to vision and NLP models. They have been applied to a wide range of fields, such as social network analysis, recommendation systems, drug discovery, physics simulation, vision-related tasks, and so on. Feature aggregation is the fundamental operation of GNNs and define how graph structure in the data is used for learning. The objective of this thesis is to propose novel GNN models with improved aggregation scheme for node ranking and node classification task. Our proposed models provide improvements over GNN models with default feature aggregation schemes and achieve higher performance than other state-of-the-art GNN models. We summarize the key contributions of this thesis in the following section.

7.1 Key Contributions

In our work, we demonstrate with experimental observations that the default aggregation scheme in graph neural networks may not be optimal for all types of different tasks. We explore alternate ways to aggregate features in the graph neural network model. The thesis begins in Chapter 1 with an introduction to graph learning and why it is becoming one of the essential and fastest-growing fields in deep learning. In Chapter 2, we provide a summary of topics that serves as the foundation of our work described in later chapters. In Chapter 4, we propose novel GNN models to approximate betweenness and closeness centrality. We achieve this by modifying the adjacency matrices so that the feature aggregation operation can approximate information flow via the shortest available paths. With the help of a novel feature aggregation scheme, our proposed model solves following the challenges related to node ranking.

- Betweenness Centrality and Closeness Centrality are computationally expensive algorithms. The current approximation algorithms face the tradeoff between ranking performance vs. execution time. With our proposed GNN based ranking models, we achieve higher ranking performance in significantly less execution time.

- Another big challenge in the ranking of nodes is the continuously changing graph structure. In real-life applications, graph structure rarely remains constant, and with time influence of nodes changes. Hence, there is a need to keep the node ranking updated with time. However, most node ranking algorithms either need to recalculate node centrality values from scratch or suffer from the inability to cope with both the addition/removal of nodes and edges. Our ranking models can provide good ranking approximations in a very short duration of time hence making them suitable for online applications with rapidly changing graphs.

In Chapter 5, we undertake a different problem of node classification and observe that the default node feature aggregation scheme is only useful under certain circumstances. We propose to learn an adaptive aggregation scheme where the model learns to select hops at which neighboring nodes provide informative features for the learning task. Our proposed method of adaptive aggregation solves the following important issues in the training of GNNs on the node classification task.

- The first problem we solve is the presence of homophily/heterophily characteristics in the datasets. Most GNN models are designed with homophily assumptions and perform poorly on heterophily datasets. Our model, due to adaptive aggregation, learns to select the best features and can perform well with both types of datasets.

- In recent years, GNN models are becoming more complex in their architecture with introduction of the attention layers, residual layers, deep networks with up to 64 layers, scaling output from each layer, and so on. With our approach, we can achieve high classification accuracy with just a simple 2-layered MLP model, which reduces model complexity and simplifies model training.

- Another major issue with graph neural networks is an over-smoothing problem where features tend to become similar after multiple hops aggregation and reduces model performance on the classification task. Our models learn to select the features that are useful with respect to label predictions and discard noisy

features. Hence, the model performs well even with 8-hop feature aggregation and solves the over-smoothing problem.

We explored two different graph learning tasks and observed that both benefited from using modified aggregation schemes specifically designed for those tasks. We studied these problems and introduced inductive biases of our understanding of the problems into the default aggregation scheme of GNNs. In the first problem, we used the shortest paths in the graph as our inductive bias, and in the second problem, we used the concepts of homophily/heterophily and the concepts of noisy features with respect to the labels to guide the design of our proposed model.

Finally, our contributions are not limited to improving GNN models on these two tasks, but they pave the way for a broader approach to modifying and adapting feature aggregation schemes for domain-specific tasks.

7.2 Future Directions

While problem-specific aggregation schemes can be tremendously helpful in model training, developing one from scratch could still be a non-trivial problem. Developing a good aggregation scheme might require sufficient domain knowledge and extensive experiments on data. Another challenge is that a good adaptive aggregation scheme for one task may not transfer well to other tasks.

It may seem that spending additional resources and time may not be a good trade-off for just using default aggregation schemes in graph neural networks. In the last decade, the field of deep learning has grown tremendously, with researchers trying to apply neural networks to all sorts of data. Most of these developments have primarily been fueled by the work of researchers in the field. However, in the last few years, some of the focus of industry and academia has also begun to develop task-oriented toolings for training deep learning models and approach the field from an engineering perspective and not just a research perspective. The efforts to develop these tools to deploy machine learning models in efficient and reliable ways (also known as Machine Learning Ops or MLOps) are approaching the problem in a domain-specific way.

With this information in mind, we can foresee the requirement of research to find ways to optimize problems in very domain-specific ways. Hence, for graph-centric data, finding aggregation schemes pertaining to a single domain can be helpful and can be made part of toolings to be used by everyone in the field.

Appendix A

Additional Information of Ranking Models

A.1 Details of Synthetic Graphs

In this section, we provide generation parameters and other relevant statistics of synthetic graphs used for evaluation of proposed models.

A.1.1 Synthetic Graph Generation Parameters

We have generated all graphs using a Python based graph library `NetworkX`[1]. Three different graph types: Erdos-Renyi (ER), Scale-Free (SF) and Gaussian Random Partition (GRP) are generated using graph generators provided in `NetworkX`. Table A.1 provides the summary of `NetworkX` functions and generation parameters used.

The notation `sample(start,end)` represents uniform sampling function which randomly samples an integer between `start` and `end`. Parameter names used are the same as provided in `NetworkX` documentation. In the case of ER graphs, when the probability value "p" is low, it is possible for the generated graph to have isolated nodes. We filter out these nodes in the pre-processing step.

A.1.2 Statistics of synthetic graphs

For evaluation of the model, 10 synthetic graphs were generated for each graph type ER, SF, and GRP. Table A.2 provides the information of synthetic datasets used as test datasets for the evaluation of our proposed model. All shortest path lengths

[1] https://networkx.github.io/

Table A.1: `NetworkX` functions and parameters used to generate synthetic ER, SF and GRP graphs

Graph Type	NetworkX function	Generation Parameters	
ER	fast_gnp_random_graph	Number of nodes	sample(50000,100000)
		p	sample(1,99)$\times 10^{-6}$
SF	scale_free_graph	Number of nodes	sample(50000,100000)
		alpha	sample(40,60)$\times 10^{-2}$
		beta	0.5
		gamma	1 - alpha - gamma
GRP	gaussian_random_partition_graph	Number of nodes	sample(50000,100000)
		s	sample(2000,10000)
		v	sample(2000,10000)
		p_in	sample(2,25)$\times 10^{-5}$
		p_out	sample(2,25)$\times 10^{-5}$

were measured using Dijkstra's algorithm and for each graph, average and maximum of all shortest path lengths are shown in the table.

A.1.3 Nodes with zero centrality values

In our proposed model, one of the important steps is to identify the nodes in the graph with no shortest paths going through them and use this information to modify the aggregation scheme in GNN-Bet and GNN-Close. Please note that such nodes in the graph will have zero betweenness centrality but not necessarily zero closeness centrality. Here we show the percentage of the nodes with both zero betweenness and closeness to provide the overview of statistics of the test graphs.

Figure A.1 & A.2, depicts the percentage of nodes which have zero betweenness and closeness centrality for synthetic and real-world evaluation datasets. It is interesting to find that the percentage of the nodes for synthetic scale-free graphs is very similar(86%-88%) . To investigate further, we generated more synthetic graphs with a wider range of generation parameters than presented in our paper. We find that for betweenness centrality, the percentage of such nodes always varied between 86%-89%. With our experimental observations, we conclude that this is an inherent property of graphs generated with `NetworkX` scale-free graph generation function.

Table A.2: Statistics of three types synthetic graphs ER, SF and GRP used for test datasets is provided.

Graph Type	**#Nodes**	**#Edges**	**Shortest Paths**	
			Avg	**Max**
ER	88,203	126,162	19.8± 15.5	91
	69,357	175,354	11.5±2.6	30
	52,369	71,773	19.6±17.2	96
	91,279	491,778	6.9±0.9	14
	73,981	173,110	12.5±3.1	35
	96,390	649,893	6.2±0.8	11
	73,311	268,817	8.6±1.3	19
	92,495	652,114	6.0±0.7	11
	92,937	612,060	6.2±0.8	12
	37,789	28,605	1.3±0.7	11
SF	85,683	144,937	4.6±2.5	17
	81,626	150,397	4.4±2.3	17
	99,689	207,035	3.5±2.1	18
	54,547	88,096	4.8±2.6	20
	70,828	154,573	4.0±1.8	13
	67,673	137,958	3.7±1.9	17
	60,137	98,727	4.3±2.8	17
	64,155	126,305	3.8±2.2	16
	83,106	136,311	4.5±2.7	18
	63,653	111,007	4.4±2.3	18
GRP	95,447	571,347	4.3± 2.7	24
	64,922	391,928	4.4±2.7	23
	50,096	188,980	2.9±1.8	17
	71,002	295,411	3.4±2.2	20
	78,081	235,093	2.5±1.6	16
	55,048	294,552	3.9±2.5	20
	86,739	371,866	3.7±2.5	21
	89,998	400,220	5.5±3.6	34
	68,483	365,743	4.2±2.7	26
	71,422	218,240	2.5±1.6	16

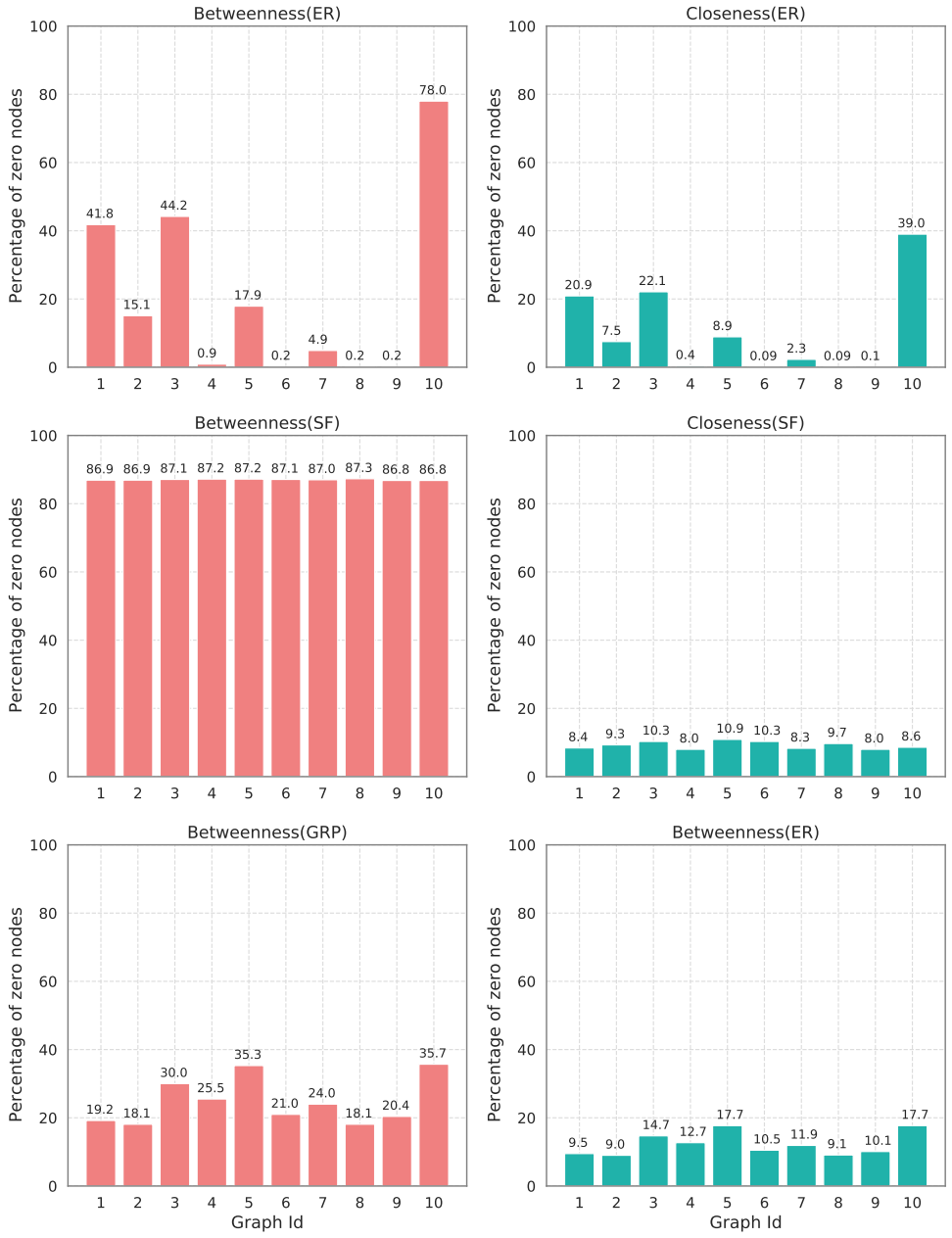


Figure A.1: Figure shows the number of nodes with zero betweenness and closeness centrality for different types of graphs in synthetic evaluation datasets

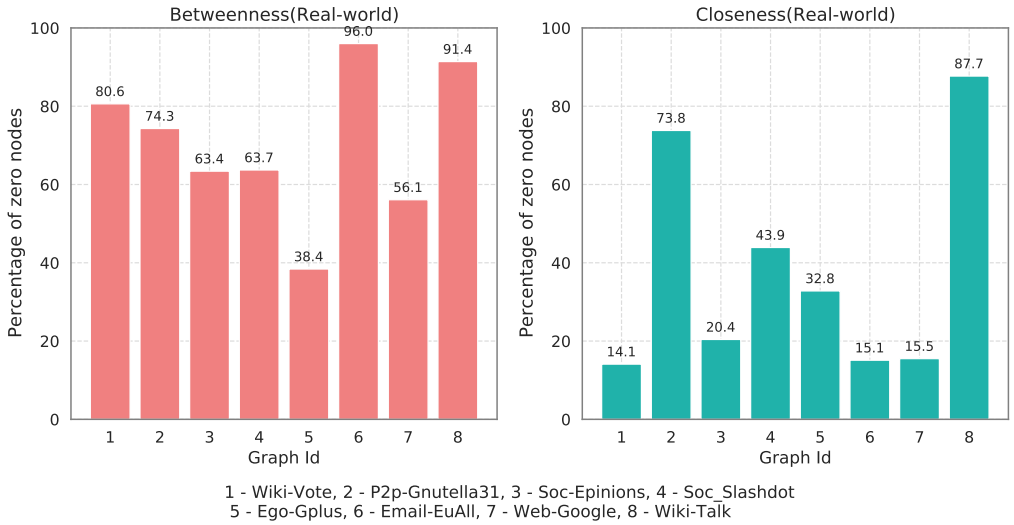


Figure A.2: Figure shows the number of nodes with zero betweenness and closeness centrality for different types of graphs in real-world datasets

A.2 Oversmoothing measurement in GNN-Bet & GNN-Close

In the current literature of GNNs, most common tasks are classification or link-prediction based tasks. In general, the performance of these types of tasks is dependent on the node having similar features to its neighbors. In conventional GNN models, a cumulative aggregation scheme is used by adding self-loops to gain and retain features of the node's neighbors at multiple hops. However, with a higher number of layers in GNN, due to this scheme, nodes tend to have similar features at the final layer, and hence it becomes difficult to learn to classify them with respect to the labels. This problem in general is known as over-smoothing in GNNs.

In our proposed scheme, we aggregate the node features at each layer separately(no self-loops) and map them to an MLP to get layer-specific scores of nodes. The scores are dependent on each node's unique neighborhood at multiple hops. Ranking loss is then calculated based on the combination of the output of the multiple layers. The shortest path lengths can go higher than seven and thus adding more information with a higher number of layers. Nevertheless, the effect diminishes due to fewer number of longer paths.

In order to empirically study the over-smoothing effect in our model, we use a metric similar to one proposed by Chen et al.[Chen et al. 2019]. In [Chen et al. 2019], the authors calculate pairwise cosine distance between node embeddings after

each layer. They observe that after each successive layer, the average cosine distance reduces thus implying presence of over-smoothing effect. To measure the effect of over-smoothing on our model's performance, we take similar approach and perform experiments on a 7-layered model. In our proposed model, the output from each layer is a score vector corresponding to the nodes in the graph. Therefore we replace cosine distance measure with the difference of two score values. As the total number of pairs of nodes can become very large ($\approx N^2$ pairs), we randomly sample 1000 non-zero centrality nodes and calculate pairwise score difference (5×10^5 node pairs) for each layer. In Table A.3 & A.4, we show the average score difference values of each layer for betweenness variant and closeness variant respectively. We observe that there is no obvious consecutive decrease in difference values with respect to the layers. Thus the model is still able to provide stable performance with 7-layers.

Table A.3: Table shows the average of pairwise score difference of nodes for each layer in 7-layered GNN-Bet model.

GNN Layers		**Average Score Difference (GNN-Bet)**		
		ER	**SF**	**GRP**
Incoming	Layer-1	0.0025 ±0.003	0.0274 ±0.028	0.0042 ±0.005
	Layer-2	0.0959 ±0.083	0.0520 ±0.043	0.0717 ±0.052
	Layer-3	0.0694 ±0.091	0.0771 ±0.060	0.0456 ±0.046
	Layer-4	0.1124 ±0.108	0.0916 ±0.072	0.1605 ±0.119
	Layer-5	0.0771 ±0.062	0.0312 ±0.028	0.0931 ±0.090
	Layer-6	0.1223 ±0.120	0.0282 ±0.025	0.1355 ±0.101
	Layer-7	0.0959 ±0.101	0.0993 ±0.109	0.0605 ±0.050
Outgoing	Layer-1	0.0025 ±0.004	0.0265 ±0.026	0.0040 ±0.004
	Layer-2	0.0956 ±0.083	0.0647 ±0.050	0.0719 ±0.052
	Layer-3	0.0688 ±0.090	0.0557 ±0.056	0.0451 ±0.046
	Layer-4	0.1120 ±0.108	0.0532 ±0.059	0.1609 ±0.119
	Layer-5	0.0768 ±0.063	0.0297 ±0.035	0.0927 ±0.089
	Layer-6	0.1232 ±0.122	0.02584 ±0.022	0.1344 ±0.100
	Layer-7	0.0970 ±0.103	0.1491 ±0.192	0.0599 ±0.049

Table A.4: Table shows the average of pairwise score difference of nodes for each layer in 7-layered GNN-Close model.

#Layer	Average Score Difference (GNN-Close)		
	ER	SF	GRP
Layer-1	0.5202 ±0.508	1.0771 ±0.892	1.9355 ±1.574
Layer-2	1.2834 ±1.052	1.5232 ±1.611	2.5324 ±2.130
Layer-3	3.1149 ±2.643	1.3551 ±1.458	3.9440 ±3.024
Layer-4	2.7950 ±2.216	1.5866 ±1.387	4.5423 ±3.724
Layer-5	0.7851 ±0.651	1.1937 ±1.133	0.9860 ±0.839
Layer-6	0.7561 ±0.886	1.2197 ±1.192	0.8559 ±0.681
Layer-7	0.8555 ±0.716	0.8492 ±0.921	1.4291 ±1.481

A.3 Ablation studies over other ranking measures

In our main experimental results, the ranking performance of the model with Kendall's Tau ranking correlation measure. Many other works in the literature [AlGhamdi et al. 2017; Fan et al. 2019a; He et al. 2022] have used this measure to evaluate ranking performance. In this section, we include more ranking measures to perform ablation studies over ranking measures.

A.3.1 Additional ranking measures

In this section we include Spearman's correlation measure [Kokoska and Zwillinger 2000] and Normalized Discounted Cumulative Gain (NDCG) [Järvelin and Kekäläinen 2002] as additional ranking measures. Table A.5 & A.6 shows the ranking scores for betweenness centrality for Spearman's correlation and NDCG correlation, respectively. Similarly, Table A.7 & A.8 show corresponding scores for closeness centrality for real-world and synthetic datasets.

A.3.2 Top-k ranking evaluation

In this section, we provide results for top-k ranking of the nodes for GNN-Bet and GNN-Closeness. We compare the performance with algorithmic methods for top-k centrality calculation. For betweenness centrality, we compare with [Borassi and Natale 2019] and for closeness centrality, we compare our model's performance with [Bergamini et al. 2019]. Results are shown in Table A.9 & A.10 for betweenness centrality and A.11 & A.12 for closeness centrality.

Table A.5: Spearman's correlation score comparison static betweenness approximation on real world and synthetic datasets.

Dataset	**Spearman rank-order Score**			
	GS	**RK**	**BN**	**GNN-Bet**
Wiki-Vote	0.998	0.960	0.921	0.997±0.0003
P2p-Gnutella31	0.995	0.982	0.915	0.994±0.0004
Soc-Epinions	0.937	0.874	0.741	0.976±0.0016
Soc-Slashdot	0.968	0.842	0.692	0.983±0.0004
Ego-Gplus	0.974	0.701	0.537	0.935±0.0032
Email-EuAll	0.968	0.513	0.383	0.999±0.0003
Web-Google	0.834	0.363	0.501	0.926±0.0013
Wiki-Talk	0.932	0.160	0.213	0.999±0.0001
Synthetic-ER	0.957±0.0417	0.944±0.0331	0.768±0.1318	0.979±0.0093
Synthetic-SF	0.620±0.0245	0.987±0.0132	0.368±0.0362	0.999±0.0001
Synthetic-GRP	0.774±0.0442	0.439±0.2212	0.180±0.1091	0.974±0.0169

Table A.6: NDCG correlation score comparison static betweenness approximation on real world and synthetic datasets.

Dataset	**NDCG Score**			
	GS	**RK**	**BN**	**GNN-Bet**
Wiki-Vote	0.997	0.999	0.997	0.944±0.0166
P2p-Gnutella31	0.995	0.998	0.990	0.910±0.0160
Soc-Epinions	0.990	0.999	0.994	0.899±0.013
Soc-Slashdot	0.988	0.998	0.993	0.917±0.0219
Ego-Gplus	0.995	0.998	0.993	0.826±0.0794
Email-EuAll	0.993	0.998	0.992	0.835±0.0526
Web-Google	0.994	0.977	0.990	0.719±0.0227
Wiki-Talk	0.991	0.933	0.959	0.894±0.0120
Synthetic-ER	0.997±0.0007	0.997±0.0021	0.985±0.0084	0.982±0.0271
Synthetic-SF	0.987±0.0132	0.998±0.0004	0.989±0.0023	0.951±0.0256
Synthetic-GRP	0.971±0.0100	0.901±0.0597	0.841±0.0377	0.980±0.0069

Table A.7: Spearman's correlation score comparison static closeness approximation on real world and synthetic datasets.

Dataset	**Spearman rank-order Score**		
	OC	**CD**	**GNN-Close**
Wiki-Vote	0.986	0.495	0.984±0.0028
P2p-Gnutella31	0.956	0.982	0.997±0.0005
Soc-Epinions	0.993	0.815	0.979±0.0014
Soc-Slashdot	0.965	0.849	0.996±0.0006
Ego-Gplus	0.994	0.935	0.971±0.0040
Email-EuAll	0.958	0.454	0.941±0.0047
Web-Google	0.984	0.488	0.839±0.0034
Wiki-Talk	0.970	0.998	0.998±0.0001
Synthetic-ER	0.939 ± 0.0010	0.724±0.2714	0.9724±0.0181
Synthetic-SF	0.952± 0.0239	0.132±0.1355	0.972±0.0127
Synthetic-GRP	0.638 ± 0.2038	-0.287±0.1540	0.997±0.0015

Table A.8: NDCG correlation score comparison static closeness approximation on real world and synthetic datasets.

Dataset	**NDCG Score**		
	OC	**CD**	**GNN-Close**
Wiki-Vote	0.999	0.894	0.997±0.0005
P2p-Gnutella31	0.999	0.935	0.998±0.0002
Soc-Epinions	0.999	0.959	0.996±0.0015
Soc-Slashdot	0.999	0.928	0.998±0.0002
Ego-Gplus	0.999	0.973	0.972±0.0012
Email-EuAll	0.999	0.938	0.998±0.0003
Web-Google	0.999	0.946	0.992±0.0014
Wiki-Talk	0.999	0.964	0.927±0.0211
Synthetic-ER	0.999 ± 0.0010	0.951±0.0552	0.998±0.0012
Synthetic-SF	0.998± 0.0012	0.908±0.0155	0.997±0.0010
Synthetic-GRP	0.946 ± 0.0482	0.750±0.0257	0.999±0.0001

Table A.9: Comparison of prediction of top-k values with [Borassi and Natale 2019] with GNN-Bet on real-world datasets. Top values are chosen as 100, 1%, 2% and 3% of total number of nodes in the graph.

Dataset		Accuracy				Time				
		Top-100	**Top-1%**	**Top-2%**	**Top-3%**	**Top-100**	**Top-1%**	**Top-2%**	**Top-3%**	**Exact**
Wiki-Vote	BN	92.0	94.3	90.8	90.6	0.02	0.02	0.03	0.03	0.8
	GNN-Bet	76.0	81.7	73.2	79.3	0.1	0.1	0.1	0.1	
P2p-Gnutella31	BN	87.0	82.8	82.1	82.8	0.1	0.2	0.2	0.2	57.2
	GNN-Bet	26.0	45.4	50.8	53.6	0.2	0.2	0.2	0.2	
Soc-Epinions	BN	94.0	91.1	88.8	87.2	0.2	0.3	0.3	0.3	222.1
	GNN-Bet	53.0	61.5	66.9	67.0	0.5	0.5	0.5	0.5	
Soc-Slashdot	BN	95.0	89.5	89.7	86.6	0.3	0.3	0.3	0.3	206.7
	GNN-Bet	52.0	70.0	78.8	81.2	0.9	0.9	0.9	0.9	
Ego-Gplus	BN	92.0	88.9	86.1	84.8	9.1	9.1	9.1	9.1	3845.7
	GNN-Bet	55.0	48.3	47.2	53.8	7.2	7.2	7.2	7.2	
Email-EuAll	BN	90.0	56.5	37.2	31.3	0.3	0.4	0.5	0.5	1091.5
	GNN-Bet	60.0	74.7	61.5	82.6	1.1	1.1	1.1	1.1	
Web-Google	BN	98.0	84.0	79.1	75.6	18.3	18.9	19.7	21.3	52422
	GNN-Bet	64.0	25.4	27.1	28.6	8.5	8.5	8.5	8.5	
Wiki-Talk	BN	81.0	50.9	47.4	47.1	0.9	1.1	1.3	1.6	32385
	GNN-Bet	61.0	73.8	70.6	72.2	3.6	3.6	3.6	3.6	

Table A.10: Comparison of prediction of top-k values with [Borassi and Natale 2019] with GNN-Bet on synthetic datasets. Top values are chosen as 100, 1%, 2% and 3% of total number of nodes in the graph.

Dataset		Average Accuracy				Average Time				
		Top-100	**Top-1%**	**Top-2%**	**Top-3%**	**Top-100**	**Top-1%**	**Top-2%**	**Top-3%**	**Exact**
Synthetic-ER	BN	40.5	30.2	48.6	28.2	0.7	0.8	0.9	0.9	811.6
	GNN-Bet	56.2	67.2	70.3	73.2	0.6	0.6	0.6	0.6	
Synthetic-SF	BN	85.4	73.1	62.8	52.3	0.04	0.1	0.1	0.1	33.8
	GNN-Bet	60.0	67.5	76.2	80.0	0.3	0.3	0.3	0.3	
Synthetic-GRP	BN	8.4	12.6	12.8	13.75	0.2	0.2	0.2	0.2	27.8
	GNN-Bet	47.1	59.8	65.2	68.2	0.6	0.6	0.6	0.6	

Table A.11: Comparison of prediction of top-k values with [Bergamini et al. 2019] with GNN-Close on real-world datasets. Top values are chosen as 100, 1%, 2% and 3% of total number of nodes in the graph.

Dataset		Accuracy				Time				
		Top-100	**Top-1%**	**Top-2%**	**Top-3%**	**Top-100**	**Top-1%**	**Top-2%**	**Top-3%**	**Exact**
Wiki-Vote	BB	100.0	100.0	100.0	100.0	0.02	0.02	0.02	0.02	0.6
	GNN-Close	81.0	84.5	83.1	81.2	0.1	0.1	0.1	0.1	
P2p-Gnutella31	BB	100.0	100.0	100.0	100.0	1.3	1.9	2.3	2.6	42.9
	GNN-Close	40.0	48.9	57.9	64.9	0.2	0.2	0.2	0.2	
Soc-Epinions	BB	100.0	99.8	100.0	100.0	1.1	2.0	2.8	3.7	160.7
	GNN-Close	60.0	73.6	77.8	80.1	0.4	0.4	0.4	0.4	
Soc-Slashdot	BB	100.0	100.0	100.0	100.0	0.5	1.2	2.0	3.2	152.7
	GNN-Close	83.0	87.9	91.1	90.1	0.5	0.5	0.5	0.5	
Ego-Gplus	BB	100.0	99.9	100.0	100.0	10.8	28.6	46.2	60.5	3458.1
	GNN-Close	20.0	45.2	55.3	63.8	6.2	6.2	6.2	6.2	
Email-EuAll	BB	100.0	100.0	96.7	100.0	0.7	7.0	13.5	16.8	807.6
	GNN-Close	81.0	76.1	69.5	57.7	0.5	0.5	0.5	0.5	
Web-Google	BB	100.0	100.0	99.9	100.0	842.0	3350.1	4477.6	5179.3	38176.7
	GNN-Close	5.0	9.6	13.6	16.1	5.3	5.3	5.3	5.3	
Wiki-Talk	BB	100.0	100.0	100.0	100.0	11.0	610.3	1107.3	1599.2	28792.0
	GNN-Close	37.0	81.4	75.0	70.6	3.1	3.1	3.1	3.1	

Table A.12: Comparison of prediction of top-k values with [Bergamini et al. 2019] with GNN-Close on synthetic datasets. Top values are chosen as 100, 1%, 2% and 3% of total number of nodes in the graph.

Dataset		Average Accuracy				Average Time				
		Top-100	**Top-1%**	**Top-2%**	**Top-3%**	**Top-100**	**Top-1%**	**Top-2%**	**Top-3%**	**Exact**
Synthetic-ER	BB	99.4	99.3	99.5	99.4	3.3	10.5	14.2	16.5	484.1
	GNN-Close	59.4	67.9	71.7	73.3	0.5	0.5	0.5	0.5	
Synthetic-SF	BB	99.2	99.3	99.5	99.4	0.4	0.5	0.6	0.6	26.4
	GNN-Close	57.5	61.5	66.6	70.1	0.2	0.2	0.2	0.2	
Synthetic-GRP	BB	100.0	100.0	100.0	100.0	0.2	0.3	0.3	0.3	28.8
	GNN-Close	58.0	70.8	76.2	80.0	0.2	0.2	0.2	0.2	

Appendix B

Additional Experiments on Node Classification

B.1 Extended experiment results

B.1.1 Node classification on higher dimensions

Table B.1 shows the mean node classification accuracy under `Single_Feature`, `All_Feature` and `Sub_Feature` settings with hidden dimensions set to 64, 128 and 256. We observe that just with increase in hidden dimensions of 2-layered MLP, we approach classification accuracy values similar or higher than state-of-the-art more complex and/or deeper GNN models.

B.1.2 Experiments with no non-linear activation

As we observe in Table B.1, with `Sub_Feature` scheme enabling feature selection, a simple 2-layered MLP already trains to very high accuracy. However, we would like to further understand the requirements of GNN complexity for the given datasets. In this section, we compare the effect of non-linear activation ReLU in 2-layered MLP model under `Sub_Feature` setting with hidden dimensions set to 256. We remove the ReLU unit between the two layers and keep other settings same for learning rate, weight decay and dropout.

Table B.2 shows the accuracy comparisons of the models with and without non-linear activation between layers. Comparing both settings, we find two interesting observations. First, for Cora, Texas and Cornell datasets, we see further improvement in accuracy values. Second, for other datasets while accuracy values have decreased

Table B.1: Mean Classification Accuracy on fully-supervised node classification task with hidden dimensions set to 64, 128 & 256.

		Single_Feature							All_feature		Sub_feature		
Dataset	**#dimensions**	**X**	**AX**	$(A+I)X$	A^2X	$(A+I)^2X$	A^3X	$(A+I)^3X$	**CAT**	**SUM**	**CAT**	**SUM**	**SOTA**
Cora	d = 64	73.40	79.55	84.28	83.86	85.47	83.58	85.41	87.68	87.5	88.10	88.04	88.49 [Chien et al. 2021; Chen et al. 2020]
	d=128	73.84	79.93	84.56	85.85	86.94	85.05	86.23	87.76	87.92	88.19	**88.43**	
	d=256	74.06	82.25	85.97	86.27	87.54	85.67	86.86	87.7	87.68	88.09	88.41	
Citeseer	d = 64	71.66	69.10	73.53	72.38	74.07	70.55	73.92	77.08	77.09	77.52	77.43	77.99 [Pei et al. 2020]
	d=128	71.94	70.74	73.96	73.70	74.58	71.42	74.28	77.35	77.04	77.70	77.63	
	d=256	72.54	71.80	76.67	75.02	76.53	72.77	75.36	77.35	77.11	**77.86**	77.74	
Pubmed	d = 64	87.79	81.77	88.27	84.70	88.06	83.06	86.63	89.75	89.55	89.88	89.83	90.30 [Chen et al. 2020]
	d=128	87.93	81.90	88.26	84.84	88.09	83.01	86.66	89.82	89.58	89.92	89.86	
	d=256	88.01	81.89	88.31	84.86	88.08	83.02	86.74	89.81	89.64	**89.97**	89.89	
Chameleon	d = 64	46.05	77.74	71.22	76.07	71.77	75.26	71.62	75.61	72.25	78.59	78.55	66.47 [Chien et al. 2021]
	d=128	46.07	77.74	71.40	76.11	71.42	76.07	71.86	75.76	71.4	78.99	77.98	
	d=256	46.09	77.63	71.25	76.77	71.07	76.2	72.58	76.77	70.81	**79.01**	77.63	
Wisconsin	d = 64	87.45	63.13	58.03	62.54	52.94	60.00	51.76	85.09	79.8	87.84	88.62	86.98 [Suresh et al. 2021]
	d=128	88.03	62.54	57.84	62.15	52.35	58.82	51.17	85.29	82.94	88.43	88.04	
	d=256	88.03	62.54	58.03	61.96	51.76	57.84	50.78	87.45	83.92	89.02	**89.22**	
Texas	d = 64	85.40	66.21	61.35	67.29	58.64	62.43	58.10	84.32	78.91	88.64	88.91	86.49 [Chien et al. 2021]
	d=128	86.21	67.02	61.62	67.56	58.64	61.62	57.83	84.32	78.91	88.38	88.11	
	d=256	85.94	67.83	61.08	67.29	58.91	61.35	58.10	86.48	82.92	**88.65**	**88.65**	
Cornell	d = 64	85.94	58.64	63.51	58.64	61.62	58.91	60.27	81.89	72.25	86.21	86.75	82.16 [Zhu et al. 2020]
	d=128	86.21	58.10	63.78	58.64	60.54	58.91	60.27	84.05	74.86	87.56	87.57	
	d=256	87.83	58.64	65.40	58.64	61.08	58.91	60.54	85.13	77.29	**88.11**	87.57	
Squirrel	d = 64	30.24	73.18	63.79	71.28	63.37	64.42	62.82	73.02	64.68	74.16	73.12	49.03 [Chien et al. 2021]
	d=128	30.30	72.83	63.66	71.49	64.43	64.49	63.59	72.55	62.50	73.87	72.78	
	d=256	30.66	72.54	63.28	71.91	65.36	65.24	63.77	72.63	59.88	**74.49**	72.76	
Actor	d = 64	35.32	25.47	29.22	25.38	27.95	25.27	26.43	35.15	35.39	35.63	35.67	36.53 [Suresh et al. 2021]
	d=128	35.75	25.38	29.26	25.25	27.71	25.26	26.21	35.94	35.57	35.96	36.05	
	d=256	36.08	25.41	29.28	25.23	27.53	25.29	26.15	36.10	35.60	36.22	**36.31**	

(expectedly), the difference is not significant except for Chameleon and Squirrel dataset.

With these results, we infer that for these datasets, simple graph convolution operation over node features combined with hop-feature selection provides sufficient information. Thus enabling a simple 2-layered model to perform well on the node classification task.

Table B.2: Mean node classification accuracy under `Sub_Feature` setting with and without using ReLU activation and hidden dimensions set to 256.

Dataset	Sub_Feature (With ReLU)		Sub_Feature (No ReLU)		SOTA
	CAT	SUM	CAT	SUM	
Cora	88.09	88.41	88.49	**88.51**	88.49
Citeseer	**77.86**	77.74	77.72	77.81	77.99
Pubmed	**89.97**	89.89	89.52	89.54	90.30
Chameleon	**79.01**	77.63	76.95	76.69	66.47
Wisconsin	89.02	**89.22**	88.63	88.82	86.98
Texas	88.65	88.65	**89.73**	88.38	86.49
Cornell	88.11	87.57	**88.38**	87.84	82.16
Squirrel	**74.49**	72.76	70.45	69.94	49.03
Actor	36.22	**36.31**	36.11	35.8	36.53

B.2 FSGNN Scalability

Table B.3: Mean classification accuracy on ogbn-100M dataset. Best performance is marked bold and second best performance is underlined.

Method	Accuracy
SGC	63.29±0.19
Node2Vec	58.07±0.28
SIGN	65.11±0.14
SAGN	66.75±0.84
GAMLP	<u>67.71±0.02</u>
FSGNN	**68.07±0.06**

In this paper, we have analyzed importance of feature selection by separating feature aggregation process with neural network component. This setting provides an additional benefit of making the model scalable. Many GNN models by design are not scalable for large graph datasets with millions of nodes as they are limited by the amount of GPU memory available in the system. To demonstrate the scalability of our model, we run experiments on `ogbn-papers100M` dataset [Wang et al. 2020][Hu et al. 2021], which is the largest public graph learning benchmark with about 111 million nodes, 1.6 billion edges and 172 node label classes. Similar to our previous experimental settings, we generate a set of features with A and $\tilde{A}$ for 4-hop aggregation. The dimension of the hidden layer is set to 1280 and γ is set to L=9 (equal to number of input features) to provide a stable training. The model is trained batchwise with input features for 10 random initializations, and we report mean accuracy.

We compare the accuracy of our model with SGC [Wu et al. 2019a], Node2Vec [Grover and Leskovec 2016], SIGN [Frasca et al. 2020], SAGN (base-model) [Sun, Gu, and Hu 2021] and GAMLP (base-model) [Zhang et al. 2021]. Similar to our method, input features can be precomputed in many of these models, thus making them scalable for larger datasets. Once features are computed, the model can be trained with small input batches of node features on the GPU. Many other GNN models cannot be trained for larger graphs as the feature generation, and model training are combined.

Table B.3 shows the mean node classification accuracy of our model along with published results of other methods taken from [Frasca et al. 2020; Hu et al. 2021; Sun, Gu, and Hu 2021; Zhang et al. 2021]. As seen from the results, FSGNN outperforms all other models. We would like to point out that classification accuracy of our model in addition to other models may be increased further by utilizing additional augmentation schemes or training measures [Li, Han, and Wu 2018; Sun, Lin, and Zhu 2020; Huang et al. 2020; Sun, Gu, and Hu 2021; Zhang et al. 2021]. However, as this is out of scope of our paper, hence we do not discuss it here.

B.3 Implementation Details of FSGNN

For reproducibility of experimental results, we provide the details of our experiment setup and hyperparameters of the model.

We use PyTorch 1.6.0 as deep learning framework on Python 3.8. Model training is done on Nvidia V100 GPU with 16 GB graphics memory and CUDA version 10.2.89.

For node classfication results (5.3), we do grid search for learning rate and weight

Table B.4: Hyperparameter search space

Hyperparameter	**Values**
WD_{sca}	0.0, 0.0001, 0.001, 0.01, 0.1
LR_{sca}	0.04, 0.02, 0.01, 0.005
WD_{fc1}	0.0, 0.0001, 0.001
WD_{fc2}	0.0, 0.0001, 0.001
LR_{fc}	0.01, 0.005
$Dropout$	0.5, 0.6, 0.7

Table B.5: Hyperparameters of the 3-hop model

Datasets	WD_{sca}	LR_{sca}	WD_{fc1}	WD_{fc2}	LR_{fc}	$Dropout$
Cora	0.1	0.01	0.001	0.0001	0.01	0.6
Citeseer	0.0001	0.005	0.001	0.0	0.01	0.5
Pubmed	0.01	0.005	0.0001	0.0001	0.01	0.7
Chameleon	0.1	0.005	0.0	0.0	0.005	0.5
Wisconsin	0.0001	0.01	0.001	0.0001	0.01	0.5
Texas	0.001	0.01	0.001	0.0	0.01	0.7
Cornell	0.0	0.01	0.001	0.001	0.01	0.5
Squirrel	0.1	0.04	0.0	0.001	0.01	0.7
Actor	0.0	0.04	0.001	0.0001	0.01	0.7

decay of the layers and dropout between the layers. Hyperparameters are set for first layer $fc1$, second layer $fc2$ and scalar weight parameter sca. ReLU is used as non-linear activation and Adam is used as the optimizer. Table B.4 shows details of hyperparameter search space. Table B.5 and B.6 show the best hyperparameters for the model in 3-hop and 8-hop configuration respectively. Patience value 100 is used for all datasets.

For experiments on ogbn-papers100M dataset, based on the data from earlier experiments, we manually tuned the hyperparameters to get the best accuracy result. Batch size of 4096 was used for training data and maximum training epoch was set to 1500. Table B.7 shows the relevant hyperparameters for the model. Patience value 300 was used in model training.

Table B.6: Hyperparameters of the 8-hop model

Datasets	WD_{sca}	LR_{sca}	WD_{fc1}	WD_{fc2}	LR_{fc}	$Dropout$
Cora	0.1	0.02	0.001	0.0001	0.01	0.6
Citeseer	0.0001	0.01	0.001	0.0001	0.01	0.5
Pubmed	0.01	0.02	0.0001	0.0	0.005	0.7
Chameleon	0.1	0.01	0.0	0.0	0.005	0.5
Wisconsin	0.001	0.02	0.001	0.0001	0.01	0.5
Texas	0.01	0.01	0.001	0.0	0.01	0.7
Cornell	0.0	0.01	0.001	0.0001	0.01	0.5
Squirrel	0.1	0.02	0.0	0.0001	0.01	0.5
Actor	0.0001	0.04	0.001	0.0001	0.01	0.7

Table B.7: Hyperparameters for the ogbn-paper100M dataset

Hyperparameters	**Values**
WD1, WD2, WD3, WDsca	1e-05, 1e-06, 9e-06, 0.1
LR1, LR2, LR3, LR_sca	5e-05, 2e-04, 2e-05, 1e-04
Dropout1, Dropout2	0.5, 0.6

Appendix C

Alternate approach to Feature Aggregation for Node Classification

In Chapter 5, we discuss how improving feature aggregation by selecting hop features is useful. Then, we propose a model FSGNN, which learns to select hops automatically in end-to-end training of the model. Taking inspiration from our previous experiments, we propose a new model that can also learn to select hop features that are most relevant to the target labels of the nodes. In this section, we provide details on the architecture of the model, and present our experiment results in comparison with other state-of-the-art GNN models and feature selection algorithms.

C.1 Feature Selection in GNN

Problems to select optimal features has been explored in machine learning literature using various feature selection strategies [Li, Chen, and Wasserman 2015; Wojtas and Chen 2020]. However, with GNNs, it is more challenging in two ways. First, node features in many datasets are sparse. For example, most datasets in Table 5.2 has more than 95% sparsity for node feature matrix. Second, the total dimensionality of node features increases rapidly with each aggregation step due to multiple feature matrices after aggregation. Due to these challenges, many feature selection algorithms are not optimal for the task. Furthermore, feature selection algorithms are used for selecting individual indices of input feature vectors. However, in the case of the node classification task, selecting hop-features is more useful as it helps to establish the similarity/dissimilarity of nodes from their neighbors. Hence it be-

comes challenging to use feature selection algorithms, and there is a need for new approaches to tackle this problem.

C.1.1 Problem Formulation

Given the input $\mathscr{X}$ as a set of $2K+1$ node feature matrices and $\mathscr{Y}$ as labels of the nodes. We are interested in identifying a subset of $\mathscr{X}$ of length up to $p < 2K+1$, for which the GNN model gives the best performance. All such possible subsets are defined as $\mathscr{M}$ and $|\mathscr{M}| = \sum_{j=1}^{p} \binom{2K+1}{j}$. Let $m \in \mathscr{M}$ denote any one of such possible subsets. Our objective is to identify the optimal subset m^* for which GNN model performs best.

C.2 Dual-Net GNN Architecture

As discussed in Section 5.3.2, the GNN model needs to have the capability to select useful node features while discarding noisy/uninformative features. In this section, we propose a new GNN model that takes all node feature matrices as input and during training learns to select optimal node feature set for the task and discards the remaining ones. Details of the model architecture and training methodology are as follows:

C.2.1 Model Description

The model consists of two separate neural networks: *classifier* network and *selector* network.

Classifier Network. It is a two-layered MLP neural network, similar to the one used in Section 5.3.1 for predicting labels of the nodes. The classifier network is parameterized with θ, represented as $f_c(\theta; \mathscr{X}, m)$. The input to the network is a subset of node feature matrices, $m \in \mathscr{M}$. In the first layer, each input matrix is linearly transformed and the output is summed. The non-linearity (ReLU) is applied and then mapped to the second layer. After training of the joint model, $f_c(\theta^*; \mathscr{X}, m^*)$ is applied on the test split of the dataset, where θ^* denotes learned parameters of the classifier and m^* is the optimal input subset of node feature matrices.

Selector Network. The selector network is also a two-layered MLP, yet with a different input/output configuration. The role of the selector is to find out an optimal feature subset of $\mathscr{X}$ based on the performance feedback of the classifier. The selector model is parameterized with ϕ, represented as $f_s(\phi, m)$. The input to the selector network is a one-hot encoding vector $\overrightarrow{m}$ of dimension $2K+1$ representing

the subset of feature matrices to be selected as per m. For example, for $K = 3$, the subset $\{X, (A+I)X, A^3X\}$ has $\overrightarrow{m} = (1, 0, 1, 0, 0, 1, 0)$. The output is a single scalar value. The purpose of the selector net is to learn to predict the average performance of the classifier to the mask corresponding to the input subset.

Both networks are trained jointly in an alternate manner. However, the training objective of both models are different.

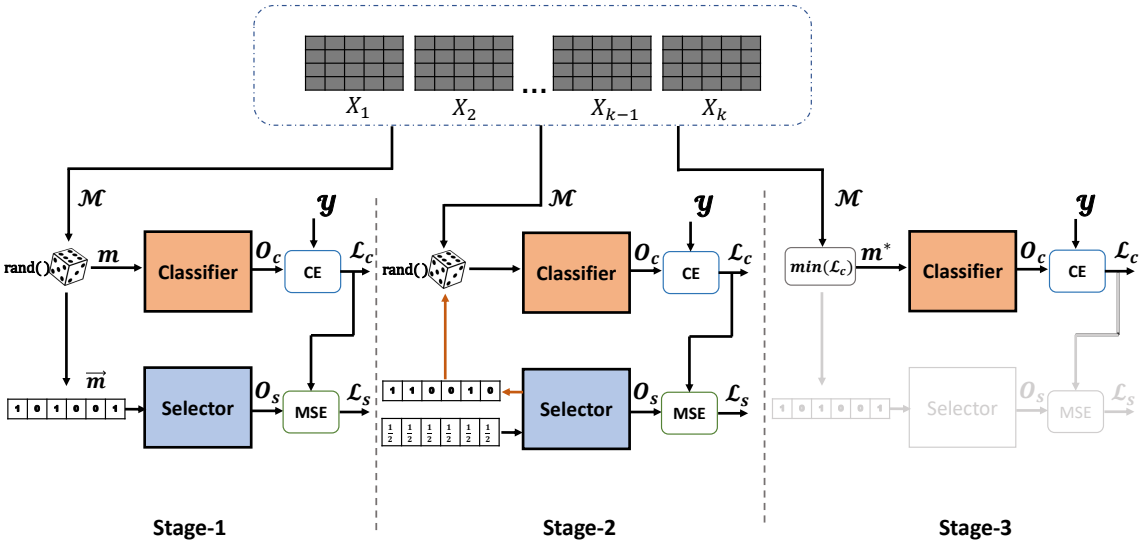


Figure C.1: Figure shows the model operation in 3-stages. Classifier and Selector are trained jointly in Stage 1 & 2, while only Classifier is trained in Stage 3.

C.2.2 Loss functions

Classifier Loss. The classifier is trained with simple Cross-Entropy loss commonly used in the node classification task. For each training step, the input is a subset of node features, $m \in \mathscr{M}$. In implementation, it represents the masking operation of input features to the model. Loss is calculated between output logits of the model and true labels of the nodes,

$$\mathscr{L}_c(\mathscr{X}, m; \theta) = \frac{1}{|V|} l(\mathscr{X}, m, y; \theta) \tag{C.1}$$

Selector Loss. The loss of selector is MSE (mean squared error) loss between the output of selector, $\mathscr{O}_s = f_s(\phi; \overrightarrow{m})$ and the classifier loss value $\mathscr{L}_c$,

$$\mathscr{L}_s(\overrightarrow{m}; \phi) = \Big(f_s(\phi; \overrightarrow{m}) - \mathscr{L}_c(\mathscr{X}, m; \theta)\Big)^2 \tag{C.2}$$

The selector learns to map the input mask vector to the classifier's loss. Thus the output of the selector varies with different input subset masks in congruence with the classifier's performance. With a good input subset, the classifier will have a lower loss, and correspondingly selector will have a lower output value and vice versa.

C.2.3 Training Methodology

Our objective in training is to start with all input feature matrices in $\mathscr{X}$, and during the process, learn the importance of each feature matrix. Based on the feature ranking, we then aim to select the optimal subset m^* to get the maximum prediction accuracy. We train our model Dual-Net GNN in three stages, with each stage having a different objective.

Stage 1. First stage is the exploration stage, where the model is trained on possible combinations of node feature matrices. The objective is to explore the performance characteristics of subsets of $\mathscr{M}$ in terms of the loss value of the classifier. For each forward pass, a random subset m sampled from $\mathscr{M}$, and corresponding node feature matrices are used as input to the classifier. Similarly, one-hot encoded vector representation $\overrightarrow{m}$ is set as input to the selector. Cross-entropy loss for the classifier is calculated and backpropagated. For the selector, MSE loss is calculated between the output of the selector $\mathscr{O}_s$ and loss of the classifier $\mathscr{L}_c$ and backpropagated to update selector network weights. The training is continued in this manner till the classifier yields different losses for different mask combinations stably, and the selector learns to map these masks to the classifier's performance. The number of training epochs in this stage is set as a hyperparameter, $E1$.

Stage 2. In this stage, our objective is to exploit the learned selector to identify indices of useful feature matrices and generate possible optimal masks for the classifier. In machine learning literature, given a trained model, gradients with respect to the input are often used to identify important input components [Ross, Hughes, and Doshi-Velez 2017; Tsang, Cheng, and Liu 2018; Hechtlinger 2016]. The input components with larger gradients contribute more toward the model's output. Similarly, we aim to identify important indices of the input mask vector to the trained selector model. The node feature matrices corresponding to these indices would lead to better performance of the classifier model.

We use a mask vector with all indices initialized with equal initial weights $(\frac{1}{2} \cdots \frac{1}{2})$ as input to the selector meaning all indices have an equal chance of being selected. We then calculate the gradients of the input vector with respect to the model. Indices with top-p gradients (i_{top-p}) are identified. It reduces the number of indices to consider from $2K + 1 \rightarrow p$. However, the optimal subset m^* might be a smaller subset of these indices, i.e. $len(m^*) \leq p$ and the number of possible candidates for optimal subset is $\sum_{j=1}^{p} \binom{p}{j}$. A fixed number of combinations of these indices are sampled, and validation loss on the classifier is calculated. Mask with the lowest loss value is selected, and both classifier and selector are trained on it for one epoch. This step is repeated each epoch, and the model is trained for a fixed number of times, $E2$.

Stage 3. After Stage 2, the input mask with the lowest validation loss is identified as m^*. The training is continued with only the classifier with node feature matrices corresponding to m^* as input. The selector model is not utilized in this stage. The training of the classifier is continued till the stopping criterion is satisfied.

C.3 Related Work

Details of related work on graph neural networks have already been discussed in Chapter 5. However, we also compare the performance of our model with feature selection methods, we provide a summary of feature selection algorithms here.

In machine learning field, feature selection has been used a way to discard noisy features to improve performance [Chandrashekar and Sahin 2014; Li et al. 2017a; Tang, Alelyani, and Liu 2014; Fukumizu and Leng 2012; Wojtas and Chen 2020]. It provides computational benefits (lower storage and faster computation) and statistical benefits including increased model interpretability [Chen et al. 2017]. There are many supervised feature selection algorithms in use. LASSO [Tibshirani 1996], random forest (RF) [Breiman 2001] , RFE (Recursive Feature Elimination) [Guyon et al. 2002] is one of the off-the-shelf method. HSIC LASSO (Hilbert Schmidt Independence Criterion Lasso)[Yamada et al. 2014] and CCM (Conditional Covariance Minimization)[Chen et al. 2017] are two kernel based methods for feature selection. Recently neural network based models [Li, Chen, and Wasserman 2016; Wojtas and Chen 2020] for feature selection have been proposed.

C.4 Experiments

In this section, we evaluate the results of experiments with our proposed model on real-world node classification datasets. We compare the performance of our model with other GNN models.

C.4.1 Settings and Baselines

For a fully-supervised node classification task, each dataset is split evenly for each class into 48%, 32%, and 20% for training, validation, and testing [Pei et al. 2020; Zhu et al. 2020]. We report the performance as mean classification accuracy over ten random splits. To provide a comprehensive evaluation of our model's performance, we compare the accuracy results with feature selection methods as well as graph neural network models.

Feature Selection Baselines. We ran experiments on 4 feature selections methods: RFE [Guyon et al. 2002], HSIC LASSO [Yamada et al. 2014], CCM [Chen et al. 2017], and DFS (Deep Feature Selection)[Li, Chen, and Wasserman 2016]. RFE, HSIC LASSO, and CCM algorithms work directly on feature matrices. Hence, we concatenated all pre-computed seven node feature matrices to get a single feature matrix, $\mathscr{X}_{cat} \in \mathbb{R}^{n \times 7d}$ and provide it as input along with labels of the nodes in the training split. For parity with our model, we set the number of dimensions that can be selected by the algorithms up to pd. Due to intensive memory requirements with high dimensional data, feature selection algorithms were run on a machine with 800 GB memory. Once the feature indices are selected by the algorithm, we trained a two-layered MLP (similar to the classifier in our model) on selected features and report the prediction accuracy. DFS is a neural network based feature selection model. We modified the DFS model to select hop features similar to our model keeping other architectural details and loss functions same.

GNN Baselines. We compare our model to 13 different GNN baselines and use the published results as the best performance of these models. GCNII [Chen et al. 2020], TDGNN [Wang and Derr 2021], H2GCN [Zhu et al. 2020] and GloGNN [Li et al. 2022] have proposed multiple variants of their model. We have chosen the variant with the best performance on most datasets. GPRGNN uses random splits in their published results. For a fair comparison, we ran their publicly available code on our standard splits while keeping other settings the same.

C.4.2 Results and Discussion

Accuracy Results

Table C.1 shows the comparison of the mean classification accuracy of our model with other feature selection methods and GNN methods. These feature selection methods overall do not perform better despite high execution time and intensive memory requirements. CCM method runs out of memory on many datasets despite using a high memory machine. The performance of DFS is comparable to a few GNN models, however, it still is significantly lower than our model. Comparing the results with other GNNs, we observe that our model overcomes the effect of noisy features and significantly improves the performance on Chameleon (+10.2%), Wisconsin (+1.6%), Texas (+4.19%), Cornell (+2.51%) and Squirrel (+27.8%) datasets. For Cora (-0.8%), Citeseer (-1.1%), Pubmed (-0.7%), and Actor (-1.1%), the performance of our model is on par with other GNN models.

Training Time

Due to its simple architecture, the training time of our model is relatively low. Figure C.2 shows a comparison of the training time of our model with a few SOTA GNN models on commonly used benchmark datasets Cora, Citeseer, and Pubmed. Training time of our model is similar to GCN and GPRGNN. Due to its deep structure with up to 64 layers, the training time of GCNII is one order magnitude higher than our model. For feature selection methods, most of the execution time is spent on running feature selection algorithm. In our experiments, feature selection algorithms took significantly higher time ($>$ 2+ hours) than graph neural network models.

C.5 Experiment Setup

C.5.1 Hardware

Experiments related feature selection algorithms (RFE, HSIC LASSO and CCM) were conducted on machine with Intel Xeon Gold 6148 processor (limited to 8 cores) and 800 GB memory. Training on feature selection algorithm, DFS and other GNN related experiments were conducted on machine with Intel Xeon Gold 6148 processor (limited to 5 cores), 60 GB memory and V100 GPU.

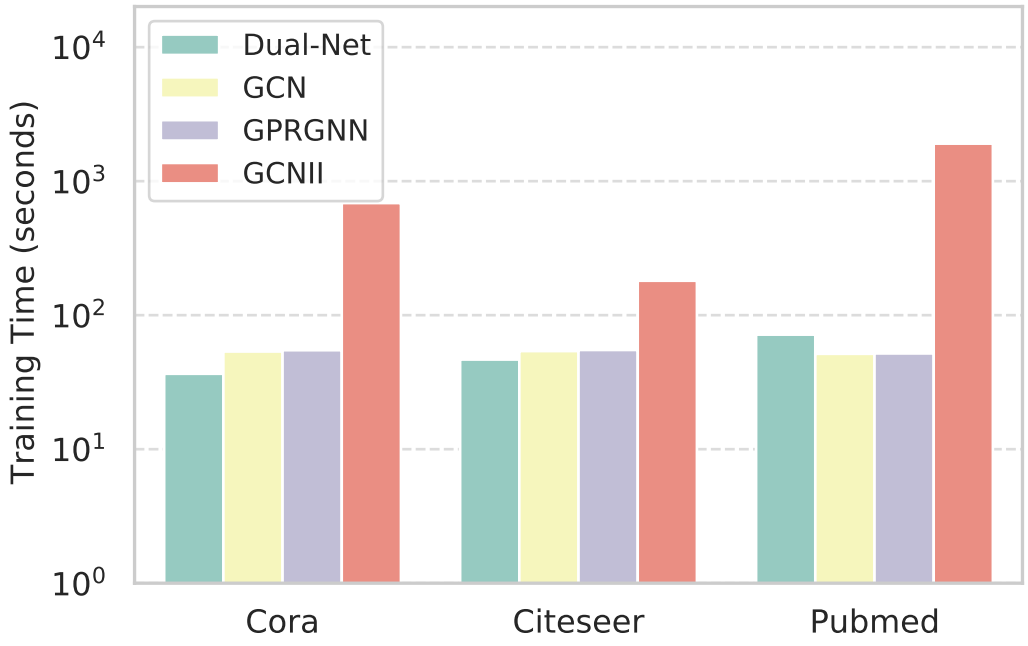


Figure C.2: Comparison of training time of GNN models on Cora, Citeseer and Pubmed datasets. Time(seconds) is plotted in log scale due to the training time of GCNII being significantly higher than other models.

C.5.2 Model Hyperparameters

For Dual-net model, the number of hops for feature aggregation, K is set to 3 and p is set as 4. Hidden dimension of both selector and classifier are set to 64. Table C.2 & C.3 lists hyperparameters related to classifier and selector model respectively. WD represents weight decay, LR represents learning rate, $fc1$ & $fc2$ denote layers and $E1$ & $E2$ are the number of epochs of training in Stage 1 and Stage 2 respectively. Patience for Stage 3 training is set to 100. For the classifier network, learning rate was selected from {0.001, 0.01, 0.05}, weight decay was selected from {0.0, 1e-05, 1e-04, 1e-03} and dropout from {0.5, 0.6, 0.7}. For the selector network, learning rate was selected from {0.01, 0.005}, weight decay was selected from {1e-06, 1e-05, 1e-04} and dropout is kept at constant value of 0.5. Training epochs $E1$ for Stage 1, were selected from 100, 200, 300, 400, 500. Search was done with mix of parameter sweep with manual intervention to avoid unfruitful hyperparameter combinations. Table C.4 shows indices of feature matrices learned for all 10 splits.

Algorithm 3: Pseudo Code Dual-Net GNN

Input : Classifier $f_c(\theta; \mathscr{X}, m)$, Selector $f_s(\phi; \overrightarrow{m})$
Input : Node feature set $\mathscr{X}$, Max. subset size p, Stage 1 epochs E_1, Stage 2 epochs E_2, hidden dimensions h
Output: Logits (Classifier)

`/* Stage-1 */`
for $ii \leftarrow 0$ **to** $E1$ **do**
  $m = Random(\mathscr{M}, i_{all}, p)$
  $\mathscr{L}_c(\mathscr{X}, m; \theta) = \frac{1}{|V|} l(\mathscr{X}, m, y; \theta)$
  $\theta'' \triangleq \theta' - \eta \nabla_\theta \mathscr{L}_c(\mathscr{X}, m; \theta)|_{\theta=\theta'}$
  $\mathscr{L}_s(\overrightarrow{m}; \phi) = \left(f_s(\phi; \overrightarrow{m}) - \mathscr{L}_c(\mathscr{X}, m; \theta) \right)^2$
  $\phi'' \triangleq \phi' - \eta \nabla_\phi \mathscr{L}_s(\overrightarrow{m}; \phi)|_{\phi=\phi'}$
end
`/* Stage-2 */`
$m_{eq} = \left(\frac{1}{2}, \frac{1}{2}, ..., \frac{1}{2} \right)$
$\delta_{m_{eq}} = \frac{\partial f_s(\phi, m)}{\partial m}|_{m=m_{eq}}$
$i_{top-p} = \text{argsort}(\delta_{m_{eq}})[-p:]$
for $jj \leftarrow 0$ **to** $E2$ **do**
  $m_p = Random(\mathscr{M}, i_{top-p}, p)$
  $\mathscr{L}_c(\mathscr{X}, m_p; \theta) = \frac{1}{|V|} l(\mathscr{X}, m_p, y; \theta)$
  $\theta'' \triangleq \theta' - \eta \nabla_\theta \mathscr{L}_c(\mathscr{X}, m_p; \theta)|_{\theta=\theta'}$
  $\mathscr{L}_s(\overrightarrow{m_p}; \phi) = \left(f_s(\phi; \overrightarrow{m_p}) - \mathscr{L}_c(\mathscr{X}, m_p; \theta) \right)^2$
  $\phi'' \triangleq \phi' - \eta \nabla_\phi \mathscr{L}_s(\overrightarrow{m_p}; \phi)|_{\phi=\phi'}$
end
`/* Stage-3 */`
$m^* = \text{argmin}(\mathscr{L}_{c(val)})$
for $kk \leftarrow 0$ **to** $E3$ **do**
  $\mathscr{L}_c(\mathscr{X}, m^*; \theta) = \frac{1}{|V|} l(\mathscr{X}, m^*, y; \theta)$
  $\theta'' \triangleq \theta' - \eta \nabla_\theta \mathscr{L}_c(\mathscr{X}, m^*; \theta)|_{\theta=\theta'}$
  if $checkEarlyStopping(\mathscr{L}_c(\mathscr{X}, m^*; \theta))$ **then**
    $\theta_t = restoreBestweights()$
    break
  end
end

Table C.1: Mean classification accuracy on fully-supervised node classification task. Results for GCN, GAT, GraphSAGE, Cheby+JK, MixHop and H2GCN-1 are taken from [Zhu et al. 2020]. For GEOM-GCN, GCNII, TDGNN, WRGAT, and GloGNN++, results are taken from the respective article. Best performance for each dataset is marked as bold. OOM means algorithm run out of memory.

	Cora	**Citeseer**	**Pubmed**	**Chameleon**	**Wisconsin**	**Texas**	**Cornell**	**Squirrel**	**Actor**	**Mean Acc.**
RFE	85.31±1.47	75.24±1.51	88.71±0.38	67.19±1.35	71.18±2.91	69.46±6.29	64.86±5.80	51.93±2.03	34.17±1.19	67.56
HSIC LASSO	84.55±2.00	73.48±1.75	89.19±0.51	63.22±2.63	79.02±3.83	79.73±8.31	77.03±7.07	44.41±1.69	33.96±1.60	69.40
CCM	85.47±1.12	OOM	OOM	67.48±1.54	72.35±3.66	70.81±4.80	64.59±5.60	OOM	OOM	-
DFS	85.31±1.47	75.24±1.51	88.71±0.38	67.19±1.35	71.18±2.91	72.16±10.18	65.95±11.85	60.11±8.05	33.79±1.94	69.93
GCN	87.28±1.26	76.68±1.64	87.38±0.66	59.82±2.58	59.80±6.99	59.46±5.25	57.03±4.67	36.89±1.34	30.26±0.79	61.62
GAT	82.68±1.80	75.46±1.72	84.68±0.44	54.69±1.95	55.29±8.71	58.38±4.45	58.92±3.32	30.62±2.11	26.28±1.73	58.55
GraphSAGE	86.90±1.04	76.04±1.30	88.45±0.50	58.73±1.68	81.18±5.56	82.43±6.14	75.95±5.01	41.61±0.74	34.23±0.99	69.50
Cheby+JK	85.49±1.27	74.98±1.18	89.07±0.30	63.79±2.27	82.55±4.57	78.38±6.37	74.59±7.87	45.03±1.73	35.14±1.37	69.89
MixHop	87.61±0.85	76.26±1.33	85.31±0.61	60.50±2.53	75.88±4.90	77.84±7.73	73.51±6.34	43.80±1.48	32.22±2.34	68.10
GEOM-GCN	85.27	**77.99**	90.05	60.90	64.12	67.57	60.81	38.14	31.63	64.05
GCNII	88.01±1.33	77.13±1.38	**90.30±0.37**	62.48±2.74	81.57±4.98	77.84±5.64	76.49±4.37	N/A	N/A	-
H2GCN-1	86.92±1.37	77.07±1.64	89.40±0.34	57.11±1.58	86.67±4.69	84.86±6.77	82.16±4.80	36.42±1.89	35.86±1.03	70.71
TDGNN-w	88.01±1.32	76.58±1.40	89.22±0.41	46.31±2.42	85.57±3.78	83.00±4.50	82.92±6.61	26.73±1.47	37.11±0.96	68.38
WRGAT	88.20±2.26	76.81±1.89	88.52±0.92	65.24±0.87	86.98±3.78	83.62±5.50	81.62±3.90	48.85±0.78	36.53±0.77	72.93
GGCN	87.95±1.05	77.14±1.45	89.15±0.37	71.14±1.84	86.86±3.29	84.86±4.55	85.68±6.63	55.17±1.58	37.54±1.56	75.05
GPRGNN	**88.49±0.95**	77.08±1.63	88.99±0.40	66.47±2.47	85.88±3.70	86.49±4.83	81.89±6.17	49.03±1.28	36.04±0.96	73.37
GloGNN++	88.33±1.09	77.22±1.78	89.24±0.39	71.21±1.84	88.04±3.22	84.05±4.90	85.95±5.81	57.88±1.76	**37.70±1.40**	75.51
Dual-Net GNN	87.77±1.16	77.15±1.58	89.64±0.43	**78.46±0.66**	**89.41±3.64**	**87.57±3.24**	**88.11±5.69**	**73.97±1.89**	37.29±0.80	**78.81**

Table C.2: Hyperparameters of the Classifier

Datasets	WD_{fc1}	WD_{fc2}	LR_{fc}	$Dropout$	$E1$	$E2$
Cora	1e-04	1e-04	0.01	0.7	400	20
Citeseer	1e-05	1e-03	0.05	0.6	500	20
Pubmed	1e-05	1e-03	0.05	0.6	400	20
Chameleon	0.0	0.0	0.001	0.6	600	20
Wisconsin	1e-04	1e-04	0.05	0.5	100	20
Texas	1e-04	1e-04	0.05	0.7	600	20
Cornell	1e-04	0.0	0.05	0.5	600	20
Squirrel	1e-04	0.0	0.001	0.5	600	20
Actor	1e-03	1e-05	0.05	0.7	600	20

Table C.3: Hyperparameters of the Selector

Datasets	WD_{both}	LR_{both}	$Dropout$
Cora	1e-05	0.005	0.5
Citeseer	1e-05	0.005	0.5
Pubmed	1e-05	0.01	0.5
Chameleon	1e-06	0.01	0.5
Wisconsin	1e-04	0.01	0.5
Texas	1e-04	0.005	0.5
Cornell	1e-04	0.005	0.5
Squirrel	1e-06	0.01	0.5
Actor	1e-04	0.01	0.5

Table C.4: Statistics of feature matrices selected in 10 random splits for each dataset

Datasets	X	AX	$(A+I)X$	A^2X	$(A+I)^2X$	A^3X	$(A+I)^3X$
Cora	6	0	2	4	3	6	10
Citeseer	10	3	5	4	8	2	6
Pubmed	10	0	1	2	6	0	3
Chameleon	0	10	4	9	1	1	1
Wisconsin	10	1	1	2	1	0	1
Texas	10	5	1	1	0	1	1
Cornell	10	2	0	1	2	2	1
Squirrel	0	10	1	3	0	2	1
Actor	10	0	0	0	0	0	0

C.6 Additional Experiments on Larger Datasets

The datasets used in our main experiments consist of graphs with a few thousand of nodes. In real-world applications, the graph size can be a few hundred of thousands of nodes to millions of nodes. Hence, the benchmarking datasets for GNN models need to be much larger than commonly used datasets. Recently [Lim et al. 2021] has proposed new benchmark datasets with large graph sizes with fixed data splits and also provided benchmark results on various GNN models. Statistics of the datasets are provided in Table C.5. To demonstrate the scalability of our model, we run experiments with Dual-Net GNN model on these datasets. Table C.6 shows the performance comparison of our model with other SOTA models. Our model, in general, outperforms other GNN models on most datasets.

Table C.5: Statistics for large graph datasets

Dataset	# Nodes	# Edges	# Feat.	# Classes
Penn94	41,554	1,362,229	5	2
pokec	1,632,803	30,622,564	65	2
arXiv-year	169,343	1,166,243	128	5
snap-patents	2,923,922	13,975,788	269	5
genius	421,961	984,979	12	2
twitch-gamers	168,114	6,797,557	7	2

C.6.1 Details of Experiments on Large Datasets

Data-Splits

For a fair comparison, we use the same data splits as provided by [Lim et al. 2021] at their code repository (https://github.com/CUAI/Non-Homophily-Large-Scale). Each dataset has five training/validation/test splits and we report mean accuracy/ ROC AUC scores in Table C.6.

Feature Calculation

We pre-calculate feature matrices with aggregation of up to 4-hops. In addition, we also include adjacency matrices as feature inputs. Due to the higher complexity of datasets, we use a 3-layered MLP model for the *Classifier*. Results for Penn94, pokec, genius, and twitch-gamers datasets are for aggregation up to 4 hops, while the results for arXiv-year and snap-patents are for up to 3 hops. Please note that

due to the large size of these datasets and longer training time, analysis similar to Table 5.1 is computationally prohibitive.

Hardware

Experiments were conducted on a machine with Intel Xeon Platinum 8360Y processor (limited to 9 cores), 60 GB memory, and A100 GPU.

Parameter Optimization

To find the best hyperparameters for the model, we use Optuna [Akiba et al. 2019], a hyperparameter optimization framework. For sampling hyperparameters during the training, we used multivariate TPESampler (tree-structured Parzen Estimator) algorithm [Bergstra et al. 2011] available in Optuna.

Table C.6: Experimental results on large node classification datasets. Mean Test accuracy is shown for most datasets, while genius shows test ROC AUC. GloGNN++ results are taken from [Li et al. 2022] and results for other models are taken from [Lim et al. 2021]. Average is taken over default five train/val/test splits. OOM denotes experiment ran out of memory. Best result for a dataset is shown in bold.

	Penn94	**pokec**	**arXiv-year**	**snap-patents**	**genius**	**twitch-gamers**	**Mean Val.**
MLP	73.61 ±0.40	62.37±0.02	36.70±0.21	31.34±0.05	86.68 ±0.09	60.92±0.07	58.60
GCN	82.47 ±0.27	75.45±0.17	46.02±0.26	45.65±0.04	87.42 ±0.37	62.18±0.26	66.53
GAT	81.53 ±0.55	71.77±6.18	46.05 ±0.51	45.37±0.44	55.80 ±0.87	59.89±4.12	60.07
GCNJK	81.63 ±0.54	77.00±0.14	46.28±0.29	46.88±0.13	89.30 ±0.19	63.45±0.22	67.42
GATJK	80.69 ±0.36	71.19±6.96	45.80 ±0.72	44.78±0.50	56.70 ±2.07	59.98±2.87	59.86
APPNP	74.33 ±0.38	62.58±0.08	38.15±0.26	32.19±0.07	85.36 ±0.62	60.97±0.10	58.93
H2GCN	OOM	OOM	49.09±0.10	OOM	OOM	OOM	-
MixHop	83.47 ±0.71	81.07±0.16	51.81±0.17	52.16±0.09	90.58 ±0.16	65.64±0.27	70.79
GPR-GNN	81.38 ±0.16	78.83±0.05	45.07±0.21	40.19±0.03	90.05 ±0.31	61.89±0.29	66.24
GCNII	82.92 ±0.59	78.94 ±0.11	47.21 ±0.28	37.88 ±0.69	90.24 ±0.09	63.39±0.61	66.76
LINKX	84.71 ±0.52	82.04±0.07	56.00 ±1.34	61.95 ±0.12	90.77 ±0.27	66.06±0.19	73.59
GloGNN++	85.74±0.42	**83.05±0.07**	54.79±0.25	62.03±0.21	90.91±0.13	66.34±0.29	73.81
Dual-Net GNN	**86.09±0.56**	81.55±0.09	**62.65±0.39**	**70.22±0.44**	**91.45±0.12**	**66.36±0.11**	**76.35**

Bibliography

Abboud, Ralph et al. (Nov. 29, 2022). *Shortest Path Networks for Graph Property Prediction*. arXiv: 2206.01003[cs].

Abu-El-Haija, Sami et al. (2019). "MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing". In: *ICML*.

Akiba, Takuya et al. (July 25, 2019). "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. New York, NY, USA: Association for Computing Machinery, pp. 2623–2631.

AlGhamdi, Ziyad et al. (June 27, 2017). "A Benchmark for Betweenness Centrality Approximation Algorithms on Large Graphs". In: *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. SSDBM '17. Chicago, IL, USA: Association for Computing Machinery, pp. 1–12.

Alon, Uri et al. (Mar. 9, 2021). *On the Bottleneck of Graph Neural Networks and its Practical Implications*. arXiv: 2006.05205[cs,stat].

Ausiello, Giorgio et al. (Jan. 7, 2017). "Directed hypergraphs: Introduction and fundamental algorithms—A survey". In: *Theoretical Computer Science*. Horn formulas, directed hypergraphs, lattices and closure systems: related formalism and application 658, pp. 293–306.

Baek, Jinheon et al. (June 28, 2021). *Accurate Learning of Graph Representations with Graph Multiset Pooling*. arXiv: 2102.11533[cs].

Banerjee, Pradeep Kr et al. (Aug. 6, 2022). *Oversquashing in GNNs through the lens of information contraction and graph expansion*. arXiv: 2208.03471[cs,math].

Bavelas, Alex (July 1, 1948). "A Mathematical Model for Group Structures". In: *Human Organization* 7.3, pp. 16–30.

— (Nov. 1, 1950). "Communication Patterns in Task-Oriented Groups". In: *The Journal of the Acoustical Society of America* 22.6, pp. 725–730.

Beauchamp, Murray A. (1965). "An improved index of centrality". In: *Behavioral Science* 10.2, pp. 161–163.

Berberidis, D. et al. (Mar. 2019). "Adaptive Diffusions for Scalable Learning Over Graphs". In: *IEEE Transactions on Signal Processing* 67.5. Conference Name: IEEE Transactions on Signal Processing, pp. 1307–1321.

Berg, Rianne van den et al. (2017). "Graph Convolutional Matrix Completion". In: *ArXiv* abs/1706.02263.

Bergamini, E. et al. (Dec. 22, 2014). "Approximating Betweenness Centrality in Large Evolving Networks". In: *2015 Proceedings of the Seventeenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. 0 vols. Proceedings. Society for Industrial and Applied Mathematics, pp. 133–146.

Bergamini, Elisabetta et al. (Sept. 24, 2019). "Computing top-k Closeness Centrality Faster in Unweighted Graphs". In: *ACM Transactions on Knowledge Discovery from Data* 13.5, 53:1–53:40.

Bergstra, James et al. (2011). "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems*. Vol. 24. Curran Associates, Inc.

Bhagat, Smriti et al. (2011). "Node Classification in Social Networks". In: *Social Network Data Analytics*. Ed. by Charu C. Aggarwal. Boston, MA: Springer US, pp. 115–148.

Bisenius, Patrick et al. (Jan. 2018). "Computing Top-k Closeness Centrality in Fully-dynamic Graphs". In: *2018 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*. Proceedings. Society for Industrial and Applied Mathematics, pp. 21–35.

Bo, Deyu et al. (2021). "Beyond Low-frequency Information in Graph Convolutional Networks". In: *AAAI*. arXiv: 2101.00797.

Borassi, Michele et al. (Feb. 2019). "KADABRA is an Adaptive Algorithm for Betweenness via Random Approximation". In: *J. Exp. Algorithmics* 24.1, 1.2:1–1.2:35.

Borgatti, Stephen P. (1995). "Centrality and AIDS". In.

— (Jan. 1, 2005). "Centrality and network flow". In: *Social Networks* 27.1, pp. 55–71.

Brandes, Ulrik (June 2001). "A faster algorithm for betweenness centrality". In: *The Journal of Mathematical Sociology* 25.2, pp. 163–177.

Brandes, Ulrik et al. (July 1, 2007). "Centrality estimation in large networks". In: *International Journal of Bifurcation and Chaos* 17.7, pp. 2303–2318.

Breiman, Leo (Oct. 1, 2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32.

Brin, S. et al. (1998). "The Anatomy of a Large-Scale Hypertextual Web Search Engine". In: Seventh International World-Wide Web Conference (WWW 1998). Brisbane, Australia.

Brockmann, Dirk et al. (Dec. 13, 2013). "The Hidden Geometry of Complex, Network-Driven Contagion Phenomena". In: *Science* 342.6164, pp. 1337–1342.

Brody, Shaked et al. (Jan. 31, 2022). *How Attentive are Graph Attention Networks?* arXiv: 2105.14491[cs].

Brohée, Sylvain et al. (Nov. 6, 2006). "Evaluation of clustering algorithms for protein-protein interaction networks". In: *BMC Bioinformatics* 7.1, p. 488.

Burges, Christopher J. C. (2010). "From RankNet to LambdaRank to LambdaMART: An Overview". In.

Cao, Nicola De et al. (2018). "MolGAN: An implicit generative model for small molecular graphs". In: *ArXiv* abs/1805.11973.

Chami, Ines et al. (2019). "Hyperbolic Graph Convolutional Neural Networks". In: *NeurIPS*.

Chandrashekar, Girish et al. (Jan. 1, 2014). "A survey on feature selection methods". In: *Computers & Electrical Engineering*. 40th-year commemorative issue 40.1, pp. 16–28.

Chen, Benson et al. (May 29, 2019). *Path-Augmented Graph Transformer Network*. arXiv: 1905.12712[cs,stat].

Chen, Deli et al. (Nov. 18, 2019). "Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View". In: *arXiv:1909.03211 [cs, stat]*. arXiv: 1909.03211.

Chen, Jianbo et al. (Dec. 4, 2017). "Kernel feature selection via conditional covariance minimization". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., pp. 6949–6958.

Chen, Jie et al. (2018). "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling". In: *ICLR*.

Chen, M. et al. (2020). "Simple and Deep Graph Convolutional Networks". In: *ICML*.

Chen, Wei et al. (2009). "Ranking Measures and Loss Functions in Learning to Rank". In: *Advances in Neural Information Processing Systems 22*. Ed. by Y. Bengio et al. Curran Associates, Inc., pp. 315–323.

Chien, Eli et al. (2021). "Adaptive Universal Generalized PageRank Graph Neural Network". In: *ICLR*.

Cohen, Edith et al. (2014). "Computing Classic Closeness Centrality, at Scale". In: *Proceedings of the Second ACM Conference on Online Social Networks*. COSN '14. New York, NY, USA: ACM, pp. 37–50.

Corso, Gabriele et al. (June 7, 2020). "Principal Neighbourhood Aggregation for Graph Nets". In: *arXiv:2004.05718 [cs, stat]*. arXiv: 2004.05718.

Crescenzi, Pierluigi et al. (Jan. 22, 2020). "Simple and Fast Distributed Computation of Betweenness Centrality". In: *arXiv:2001.08108 [cs]*. arXiv: 2001.08108.

Defferrard, Michaël et al. (2016). "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *arXiv:1606.09375*.

Ding, Ying et al. (Nov. 2009). "PageRank for Ranking Authors in Co-citation Networks". In: *J. Am. Soc. Inf. Sci. Technol.* 60.11, pp. 2229–2243.

Dosovitskiy, Alexey et al. (June 3, 2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv: 2010.11929[cs].

Duvenaud, David K et al. (2015). "Convolutional Networks on Graphs for Learning Molecular Fingerprints". In: *NIPS*, pp. 2224–2232.

Edmonds, Nick et al. (Dec. 2010). "A space-efficient parallel algorithm for computing betweenness centrality in distributed memory". In: *2010 International Conference on High Performance Computing*. 2010 International Conference on High Performance Computing. ISSN: 1094-7256, pp. 1–10.

Eppstein, David et al. (2001). "Fast Approximation of Centrality". In: *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '01. event-place: Washington, D.C., USA. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, pp. 228–229.

Erciyes, Kayhan (2013). *Distributed Graph Algorithms for Computer Networks*. Springer Publishing Company, Incorporated.

Errica, Federico et al. (2020). "A Fair Comparison of Graph Neural Networks for Graph Classification". In: *ICLR*.

Fan, Angela et al. (Jan. 1, 2021). "Beyond english-centric multilingual machine translation". In: *The Journal of Machine Learning Research* 22.1, 107:4839–107:4886.

Fan, Changjun et al. (2019a). "Learning to Identify High Betweenness Centrality Nodes from Scratch: A Novel Graph Neural Network Approach". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. event-place: Beijing, China. New York, NY, USA: ACM, pp. 559–568.

Fan, Wenqi et al. (May 13, 2019b). "Graph Neural Networks for Social Recommendation". In: *The World Wide Web Conference*. WWW '19. New York, NY, USA: Association for Computing Machinery, pp. 417–426.

Feng, Wenzheng et al. (2020). "Graph Random Neural Networks for Semi-Supervised Learning on Graphs". In: *Advances in Neural Information Processing Systems* 33.

Ferber, C. von et al. (Mar. 1, 2009). "Public transport networks: empirical analysis and modeling". In: *The European Physical Journal B* 68.2, pp. 261–275.

Franceschet, Massimo (June 2011). "PageRank: Standing on the Shoulders of Giants". In: *Commun. ACM* 54.6, pp. 92–101.

Frasca, Fabrizio et al. (2020). "SIGN: Scalable Inception Graph Neural Networks". In: *arXiv:2004.11198 [cs, stat]*.

Freeman, Linton C. (1977). "A Set of Measures of Centrality Based on Betweenness". In: *Sociometry* 40.1, pp. 35–41.

— (Jan. 1, 1978). "Centrality in social networks conceptual clarification". In: *Social Networks* 1.3, pp. 215–239.

Fukumizu, K. et al. (2012). "Gradient-based kernel method for feature extraction and variable selection". In: *NIPS*.

Geisberger, Robert et al. (2008). "Better Approximation of Betweenness Centrality". In: *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pp. 90–100.

Gilmer, Justin et al. (2017). "Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17.

Gkantsidis, C. et al. (2003). "The Markov Chain Simulation Method for Generating Connected Power Law Random Graphs". In: Workshop on Algorithm Engineering and Experimentation.

Gleich, David F. (Jan. 1, 2015). "PageRank Beyond the Web". In: *SIAM Review* 57.3, pp. 321–363.

Godwin, Jonathan et al. (2021). "Very Deep Graph Neural Networks Via Noise Regularisation". In: *arXiv:2106.07971 [cs]*.

Green, O. et al. (Sept. 2012). "A Fast Algorithm for Streaming Betweenness Centrality". In: *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*. 2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing, pp. 11–20.

Grover, Aditya et al. (2016). "node2vec: Scalable Feature Learning for Networks". In: *KDD*.

Guo, Zhijiang et al. (July 2019). "Attention Guided Graph Convolutional Networks for Relation Extraction". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. ACL 2019. Florence, Italy: Association for Computational Linguistics, pp. 241–251.

Guyon, Isabelle et al. (Jan. 1, 2002). "Gene Selection for Cancer Classification using Support Vector Machines". In: *Machine Learning* 46.1, pp. 389–422.

Hagberg, Aric A. et al. (2008). "Exploring network structure, dynamics, and function using NetworkX". In: *In Proceedings of the 7th Python in Science Conference (SciPy*, pp. 11–15.

Hamilton, Will et al. (2017). "Inductive Representation Learning on Large Graphs". In: *NIPS*, pp. 1024–1034.

Hayashi, Takanori et al. (2015). "Fully Dynamic Betweenness Centrality Maintenance on Massive Networks". In: *PVLDB* 9, pp. 48–59.

He, Yixuan et al. (June 28, 2022). "GNNRank: Learning Global Rankings from Pairwise Comparisons via Directed Graph Neural Networks". In: *Proceedings of the 39th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, pp. 8581–8612.

Hechtlinger, Yotam (Nov. 22, 2016). "Interpretation of Prediction Models Using the Input Gradient". In: *arXiv:1611.07634 [cs, stat]*. arXiv: 1611.07634.

Hoang, Loc et al. (Feb. 16, 2019). "A round-efficient distributed betweenness centrality algorithm". In: *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*. PPoPP '19. Washington, District of Columbia: Association for Computing Machinery, pp. 272–286.

Hu, Weihua et al. (2019). "Pre-training Graph Neural Networks". In: *ArXiv*.

Hu, Weihua et al. (Jan. 23, 2021). "Open Graph Benchmark: Datasets for Machine Learning on Graphs". In: *arXiv:2005.00687 [cs, stat]*. version: 4. arXiv: 2005.00687.

Huang, Qian et al. (Nov. 2, 2020). "Combining Label Propagation and Simple Models Out-performs Graph Neural Networks". In: *arXiv:2010.13993 [cs]*. arXiv: 2010.13993.

Jin, S. et al. (Apr. 2010). "A novel application of parallel betweenness centrality to power grid contingency analysis". In: *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*. 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS), pp. 1–7.

Jin, Wei et al. (Mar. 8, 2021). "Node Similarity Preserving Graph Convolutional Networks". In: WSDM '21. Association for Computing Machinery, pp. 148–156.

Jin, Wengong et al. (Jan. 28, 2019). "Learning Multimodal Graph-to-Graph Translation for Molecular Optimization". In: *arXiv:1812.01070 [cs, stat]*. arXiv: 1812.01070.

Joy, Maliackal Poulo et al. (2005). "High-Betweenness Proteins in the Yeast Protein Interaction Network". In: *Journal of Biomedicine and Biotechnology* 2005.2, pp. 96–103.

Järvelin, Kalervo et al. (Oct. 1, 2002). "Cumulated gain-based evaluation of IR techniques". In: *ACM Transactions on Information Systems* 20.4, pp. 422–446.

Kas, M. et al. (Aug. 2013). "Incremental algorithm for updating betweenness centrality in dynamically growing networks". In: *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*. 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013), pp. 33–40.

Kendall, M. G. (June 1, 1938). "A NEW MEASURE OF RANK CORRELATION". In: *Biometrika* 30.1, pp. 81–93.

Kipf, Thomas N. et al. (2017). "Semi-Supervised Classification with Graph Convolutional Networks". In: *ICLR*. arXiv: 1609.02907.

Klicpera, Johannes et al. (2018). *Predict then Propagate: Combining neural networks with personalized pagerank for classification on graphs*.

Kokoska, Stephen et al. (Mar. 29, 2000). *CRC Standard Probability and Statistics Tables and Formulae, Student Edition*.

Kourtellis, N. et al. (May 2016). "Scalable online betweenness centrality in evolving graphs". In: *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp. 1580–1581.

Kuang, Da et al. (Apr. 26, 2012). "Symmetric Nonnegative Matrix Factorization for Graph Clustering". In: *Proceedings of the 2012 SIAM International Conference on Data Mining (SDM)*. Proceedings. Society for Industrial and Applied Mathematics, pp. 106–117.

Lee, Kwang Hee et al. (Mar. 13, 2020). "Linearization of Dependency and Sampling for Participation-based Betweenness Centrality in Very Large B-hypergraphs". In: *ACM Transactions on Knowledge Discovery from Data* 14.3, 25:1–25:41.

Lee, Min-Joong et al. (2012). "QUBE: a quick algorithm for updating betweenness centrality". In: *WWW*.

Lee, Youngwan et al. (June 2020). "CenterMask: Real-Time Anchor-Free Instance Segmentation". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). ISSN: 2575-7075, pp. 13903–13912.

Li, Guohao et al. (2021). "Training Graph Neural Networks with 1000 Layers". In: *ICML*.

Li, Jundong et al. (Dec. 6, 2017a). "Feature Selection: A Data Perspective". In: *ACM Computing Surveys* 50.6, 94:1–94:45.

Li, Qimai et al. (Jan. 22, 2018). "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning". In: *arXiv:1801.07606 [cs, stat]*. arXiv: 1801.07606.

Li, Xiang et al. (May 15, 2022). *Finding Global Homophily in Graph Neural Networks When Meeting Heterophily*. arXiv:2205.07308. type: article. arXiv. arXiv: 2205.07308[cs].

Li, Yifeng et al. (2015). "Deep Feature Selection: Theory and Application to Identify Enhancers and Promoters". In: *Research in Computational Molecular Biology*. Ed. by Teresa M. Przytycka. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 205–217.

— (May 2016). "Deep Feature Selection: Theory and Application to Identify Enhancers and Promoters". In: *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology* 23.5, pp. 322–336.

Li, Yujia et al. (Sept. 22, 2017b). *Gated Graph Sequence Neural Networks*. arXiv: 1511.05493[cs,stat].

Li, Yujia et al. (2018). "Learning Deep Generative Models of Graphs". In: *ArXiv*. ICLR.

Li, Yujia et al. (2019). "Graph Matching Networks for Learning the Similarity of Graph Structured Objects". In: *ICML*.

Liao, Hao et al. (May 19, 2017). "Ranking in evolving complex networks". In: *Physics Reports*. Ranking in evolving complex networks 689, pp. 1–54.

Lim, Derek et al. (Oct. 27, 2021). *Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods*. arXiv: 2110.14446[cs,stat].

Liu, Yang et al. (June 3, 2021). "Pick and Choose: A GNN-based Imbalanced Learning Approach for Fraud Detection". In: *Proceedings of the Web Conference 2021*. WWW '21. New York, NY, USA: Association for Computing Machinery, pp. 3168–3177.

Loukas, Andreas (2020). "What graph neural networks cannot learn: depth vs width". In: *ICLR*.

László, Lovász (Jan. 1, 1996). "Random Walks on Graphs: A Survey". In: Combinatorica, pp. 1–46.

Ma, Yao et al. (2021). "Is Homophily a Necessity for Graph Neural Networks?" In: *arXiv:2106.06134 [cs, stat]*.

Madhawa, Kaushalya et al. (2019). "GraphNVP: An Invertible Flow Model for Generating Molecular Graphs". In: *arXiv:1905.11600*.

Marcheggiani, Diego et al. (Sept. 2017). "Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling". In: EMNLP 2017. ACL, pp. 1506–1515.

Maron, Haggai et al. (2019). "Provably Powerful Graph Networks". In: *ArXiv* abs/1905.11136.

Maurya, Sunil K. et al. (2019). "Fast Approximations of betweenness centrality using Graph Neural Networks". In: International Conference on Information and Knowledge Management (CIKM).

Maurya, Sunil Kumar et al. (May 10, 2021). "Graph Neural Networks for Fast Node Ranking Approximation". In: *ACM Transactions on Knowledge Discovery from Data* 15.5, 78:1–78:32.

— (2022a). "Not All Neighbors are Friendly: Learning to Choose Hop Features to Improve Node Classification". In: International Conference on Information and Knowledgement (CIKM).

— (July 1, 2022b). "Simplifying approach to node classification in Graph Neural Networks". In: *Journal of Computational Science* 62, p. 101695.

McPherson, M. et al. (2001). "Birds of a Feather: Homophily in Social Networks". In.

Medo, Matúš (2013). "Network-Based Information Filtering Algorithms: Ranking and Recommendation". In: *Dynamics On and Of Complex Networks, Volume 2: Applications to Time-Varying Dynamical Systems*. Ed. by Animesh Mukherjee et al. Modeling and Simulation in Science, Engineering and Technology. New York, NY: Springer New York, pp. 315–334.

Mika, Peter (2004). "Social Networks and the Semantic Web". In: *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*. WI '04. Washington, DC, USA: IEEE Computer Society, pp. 285–291.

Mislove, Alan et al. (Oct. 24, 2007). "Measurement and analysis of online social networks". In: *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. IMC '07. New York, NY, USA: Association for Computing Machinery, pp. 29–42.

Monge, Peter R. et al. (2003). *Theories of Communication Networks*. Google-Books-ID: 5z3oPq8M5NwC. Oxford University Press. 434 pp.

Morris, Christopher et al. (2018). "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks". In: *AAAI*.

Mouton, Coenraad et al. (2020). "Stride and Translation Invariance in CNNs". In: *Artificial Intelligence Research*. Ed. by Aurona Gerber. Communications in Computer and Information Science. Cham: Springer International Publishing, pp. 267–281.

Newman, M. E. J. (Jan. 1, 2005). "A measure of betweenness centrality based on random walks". In: *Social Networks* 27.1, pp. 39–54.

Newman, Mark (2010). *Networks: An Introduction*. New York, NY, USA: Oxford University Press, Inc.

NT, Hoang et al. (2020). "Stacked Graph Filter". In: *arXiv:2011.10988 [cs]*.

Okamoto, Kazuya et al. (2008). "Ranking of Closeness Centrality for Large-Scale Social Networks". In: *Frontiers in Algorithmics*. Ed. by Franco P. Preparata et al. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 186–195.

Oono, Kenta et al. (2020). "Graph Neural Networks Exponentially Lose Expressive Power for Node Classification". In: *ICLR*.

Ortega, Antonio et al. (May 2018). "Graph Signal Processing: Overview, Challenges, and Applications". In: *Proceedings of the IEEE* 106.5. Conference Name: Proceedings of the IEEE, pp. 808–828.

Papp, Pál András et al. (Nov. 11, 2021). *DropGNN: Random Dropouts Increase the Expressiveness of Graph Neural Networks*. arXiv: 2111.06283[cs].

Parisot, Sarah et al. (2017). "Spectral Graph Convolutions for Population-Based Disease Prediction". In: *Medical Image Computing and Computer Assisted Intervention, MICCAI 2017*. Ed. by Maxime Descoteaux et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 177–185.

Paszke, Adam et al. (2017). "Automatic differentiation in PyTorch". In: *NIPS Autodiff Workshop*, p. 4.

Pei, Hongbin et al. (2020). "Geom-GCN: Geometric Graph Convolutional Networks". In: *ICLR*.

Pei, Sen et al. (July 3, 2014). "Searching for superspreaders of information in real-world social media". In: *Scientific Reports* 4, p. 5547.

Pfaff, Tobias et al. (June 18, 2021). "Learning Mesh-Based Simulation with Graph Networks". In: *arXiv:2010.03409 [cs]*. arXiv: 2010.03409.

Pham, M. C. et al. (Aug. 2010). "The Structure of the Computer Science Knowledge Network". In: *2010 International Conference on Advances in Social Networks Analysis and Mining*. 2010 International Conference on Advances in Social Networks Analysis and Mining, pp. 17–24.

Puzis, Rami et al. (2012). "Augmented Betweenness Centrality for Environmentally Aware Traffic Monitoring in Transportation Networks". In: *J. Intellig. Transport. Systems* 17, pp. 91–105.

Rakhimberdina, Zarina et al. (2020). "Linear Graph Convolutional Model for Diagnosing Brain Disorders". In: *Complex Networks and Their Applications VIII*. Ed. by Hocine Cherifi et al. Studies in Computational Intelligence. Cham: Springer International Publishing, pp. 815–826.

Ramesh, Aditya et al. (July 1, 2021). "Zero-Shot Text-to-Image Generation". In: *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, pp. 8821–8831.

Riondato, Matteo et al. (Mar. 1, 2016). "Fast approximation of betweenness centrality through sampling". In: *Data Mining and Knowledge Discovery* 30.2, pp. 438–475.

Rong, Y. et al. (2020). "DropEdge: Towards Deep Graph Convolutional Networks on Node Classification". In: *ICLR*.

Ross, A. et al. (2017). "Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations". In: *IJCAI*.

Rozemberczki, Benedek et al. (Mar. 10, 2020). "Multi-scale Attributed Node Embedding". In: *arXiv:1909.13021 [cs, stat]*.

Sabidussi, Gert (Dec. 1, 1966). "The centrality index of a graph". In: *Psychometrika* 31.4, pp. 581–603.

Sen, P. et al. (2008). "Collective Classification in Network Data". In: *AI Mag.*

Srivastava, Nitish et al. (Jan. 1, 2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Staudt, Christian L. et al. (Dec. 2016). "NetworKit: A tool suite for large-scale complex network analysis". In: *Network Science* 4.4, pp. 508–530.

Sun, Chuxiong et al. (July 1, 2021). "Scalable and Adaptive Graph Neural Networks with Self-Label-Enhanced training". In: *arXiv:2104.09376 [cs]*. arXiv: 2104.09376.

Sun, Ke et al. (Feb. 20, 2020). "Multi-Stage Self-Supervised Learning for Graph Convolutional Networks on Graphs with Few Labels". In: *arXiv:1902.11038 [cs, stat]*. arXiv: 1902.11038.

Suresh, Susheel et al. (June 11, 2021). "Breaking the Limit of Graph Neural Networks by Improving the Assortativity of Graphs with Local Mixing Patterns". In: *arXiv:2106.06586 [cs]*.

Tang, Jie et al. (June 28, 2009). "Social influence analysis in large-scale networks". In: KDD '09. Association for Computing Machinery, pp. 807–816.

Tang, Jiliang et al. (Jan. 1, 2014). "Feature selection for classification: A review". In: *Data Classification: Algorithms and Applications*. Publisher: CRC Press, pp. 37–64.

Tibshirani, Robert (1996). "Regression Shrinkage and Selection Via the Lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1996.tb02080.x, pp. 267–288.

Tsang, Michael et al. (Feb. 27, 2018). "Detecting Statistical Interactions from Neural Network Weights". In: *arXiv:1705.04977 [cs, stat]*. arXiv: 1705.04977.

Vaswani, Ashish et al. (Dec. 5, 2017). *Attention Is All You Need*. arXiv: 1706.03762[cs].

Velickovic, Petar et al. (2018a). "Deep Graph Infomax". In: ICLR.

Velickovic, Petar et al. (2018b). "Graph Attention Networks". In: *ICLR*.

Wang, Kuansan et al. (Jan. 23, 2020). "Microsoft Academic Graph: When experts are not enough". In: *Quantitative Science Studies* 1.1. Publisher: MIT Press, pp. 396–413.

Wang, Yining et al. (June 13, 2013). "A Theoretical Analysis of NDCG Type Ranking Measures". In: *Proceedings of the 26th Annual Conference on Learning Theory*. Conference on Learning Theory. ISSN: 1938-7228. PMLR, pp. 25–54.

Wang, Yu et al. (Aug. 24, 2021). "Tree Decomposed Graph Neural Network". In: *CIKM*. arXiv: 2108.11022.

Wang, Zhen et al. (Dec. 9, 2016). "Statistical physics of vaccination". In: *Physics Reports*. Statistical physics of vaccination 664, pp. 1–113.

Wojtas, Maksymilian et al. (Oct. 18, 2020). "Feature Importance Ranking for Deep Learning". In: *arXiv:2010.08973 [cs]*. arXiv: 2010.08973.

Woo, Sanghyun et al. (Dec. 3, 2018). "LinkNet: relational embedding for scene graph". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., pp. 558–568.

Wu, Felix et al. (2019a). "Simplifying Graph Convolutional Networks". In: *ICML*. arXiv: 1902.07153.

Wu, Shiwen et al. (Apr. 19, 2021). "Graph Neural Networks in Recommender Systems: A Survey". In: *arXiv:2011.02260 [cs]*. arXiv: 2011.02260.

Wu, Zonghan et al. (Dec. 3, 2019b). "A Comprehensive Survey on Graph Neural Networks". In: *arXiv:1901.00596 [cs, stat]*.

Xia, Feng et al. (Apr. 2021). "Graph Learning: A Survey". In: *IEEE Transactions on Artificial Intelligence* 2.2. Conference Name: IEEE Transactions on Artificial Intelligence, pp. 109–127.

Xu, Keyulu et al. (July 3, 2018). "Representation Learning on Graphs with Jumping Knowledge Networks". In: ICML. PMLR.

Xu, Keyulu et al. (2019). "How Powerful are Graph Neural Networks?" In: International Conference on Learning Representation (ICLR). arXiv: 1810.00826.

Yamada, Makoto et al. (Jan. 1, 2014). "High-dimensional feature selection by feature-wise kernelized Lasso". In: *Neural Computation* 26.1, pp. 185–207.

Yang, Jianwei et al. (2018). "Graph R-CNN for Scene Graph Generation". In: *Computer Vision – ECCV 2018*. Ed. by Vittorio Ferrari et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 690–706.

Yin, Yongjing et al. (2020). "A Novel Graph-based Multi-modal Fusion Encoder for Neural Machine Translation". In: *ACL*.

Ying, Rex et al. (2018a). "Graph Convolutional Neural Networks for Web-Scale Recommender Systems". In: *KDD '18*.

Ying, Rex et al. (2018b). "Hierarchical Graph Representation Learning with Differentiable Pooling". In: NIPS'18, pp. 4805–4815.

Yun, Seongjun et al. (Feb. 4, 2020). *Graph Transformer Networks*. arXiv: 1911.06455[cs,stat].

Zhang, Muhan et al. (2018). "An End-to-End Deep Learning Architecture for Graph Classification". In: *AAAI*.

Zhang, Wentao et al. (Sept. 9, 2021). "Graph Attention Multi-Layer Perceptron". In: *arXiv:2108.10097 [cs]*. arXiv: 2108.10097.

Zhang, Z. et al. (2020). "Deep Learning on Graphs: A Survey". In: *IEEE Transactions on Knowledge and Data Engineering*. Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 1–1.

Zhou, Jie et al. (July 10, 2019). "Graph Neural Networks: A Review of Methods and Applications". In: *arXiv:1812.08434 [cs, stat]*. arXiv: 1812.08434.

Zhu, Jiong et al. (2020). "Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs". In: *NeurIPS* 33.

Zhu, Jiong et al. (2021). "Graph Neural Networks with Heterophily". In: *AAAI*.