Geometric Deep Learning: Impact of Graph Structure on Graph Neural Networks

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Mark Cheung

B.S., Electrical and Computer Engineering, University of Viginia
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

September 2022

Acknowledgements

I am extremely grateful to my advisor, Prof. José Moura, for his invaluable guidance, unwavering support, and patience during my PhD study. I would also like to express my gratitude to Prof. David Danks, Prof. Soummya Kar, and Prof. Giulia Fanti for serving on my committee and providing comments that helped enhance this thesis.

I am fortunate to have a great set of collaborators and colleagues including Jonathan Mei, Yang Li, John Shi, Oren Wright, Lavender Jiang, Chris Liu, Wendy Summer, and Austin Lin. Also, thanks to the past and present members of my group, Carol Patterson, Rajshekhar Das, Srinivasa Pranav, Shreyas Chaudhari, Kawisorn Kamtue, Joya Deri, Evgeny Toropov, Satwik Kottur, Stephen Kruzick, Subhro Das, Chaojing Duan, Tong Shen, Paola Buitrago, Wei Li, June Zhang, Lynna Gui, Shanghang Zhang, Yuan Chen, Umang Bhatt, Jian Du, Yixiong Zou and others. Additionally, I would like to thank Nathan Snizaski, Claire Bauerle, Sherri Ferris, Christina Cowan, Tara Moe, Greta Ruperto, Megan Oliver, and many other members of the ECE department staff for their support with administrative tasks.

There are many people that have made me feel at home in Pittsburgh, including CMU and University of Pittsburgh meditation/mindfulness groups (Sophia Zhu), former classmates (Josue Orellana, Lingxiao Zhao), previous roommates and landlords (Will Halim, Sheila Smith), faculty and staff (Prof. Robert Kass, Pattye Stragar). There are too many to list here.

I could not have gotten to this point without the love and support from my parents Kee Nam Cheung and Anna Cheung, my sister Connie Cheung, her fiance Varun Rishi, and all of my extended family. I would also to like to dedicate my thesis to my beloved teacher Ishwar Puri who passed away in 2020. His love is always with me.

My Ph.D. was supported in part by a Carnegie Institute of Technology Dean Fellowship, the Department of Defense Contract No. FA8702-15-D-0002, and the National Science Foundation grants CPS 1837607 and CIF 1513936.

Abstract

Deep learning techniques have led to major improvements in fields like natural language processing, computer vision, and other Euclidean data domains, yet in many domains data are irregular, requiring graphs or manifolds to be explicitly modeled. Such applications include social networks, sensor feeds, logistics, supply chains, chemistry, neuroscience, and other biological systems. The extension of deep learning to these non-Euclidean data is an area of research now called Geometric Deep Learning (GDL). This thesis focuses on a subfield of GDL, graph neural networks (GNNs) that learn on graph signals using neural networks. We explore the impact of the data graph structure on the performance of graph neural networks using real and synthetic data for two graph learning tasks: node and graph classification.

We start with the formalization of GNNs, and consider two flavors of approaches: spectral approach typified by graph convolutional networks (GCNs) and spatial approach typified by topology adaptive graph convolutional networks (TAGCNs). In general, TAGCN requires fewer number of layers than GCN, with moderate degrees of the polynomial filters.

For node classification, not many layers are needed to achieve optimal performance. Unlike graph classification, graph signal is necessary and important. For some real datasets, classifying using a simple estimator on the graph signals can outperform GNNs. For synthetic datasets, Erdős-Rényi and preferential attachment models have similar test accuracy curves for both GCN and TAGCN with respect to the number of layers and the degree of the polynomial filters. For smallworld model, TAGCN's filters play an important role in achieving the optimal accuracy and accelerating the effect of over-smoothing.

We also study the training convergence for node classification. We show theoretically that training loss converges to a global minimum for linearized TAGCN. Despite the non-convex objectives, training loss for 1-degree H-layer TAGCN, i.e., with degree

1 polynomial filter and H layers, is guaranteed to converge to global minimum at an exponential rate, faster with higher number of layers. With K-degree TAGCN, convergence is accelerated with higher degree K of the polynomial filters. We experimentally validate our theory and show training convergence holds true for both linearized and nonlinearized TAGCN.

For graph classification, graph structure plays a more important role than the graph signal. GNNs can often classify graphs using just the graph structures of the different classes if they are distinct enough. For real datasets, we relate simple network metrics and signal statistics to the performance of these models. We show for some datasets, classifiers on number of edges or number of nodes can lead to better or similar performance as graph neural networks. For other datasets, signal statistics can perform well. Based on these observations, we were able to apply simple modifications to GCN and TAGCN to improve their performance (sumpool and degree-aware TAGCN). For synthetic datasets, Erdős-Rényi and preferential attachment models again have similar test accuracy curves for both GCN and TAGCN. For smallworld model, more than 1 layer is needed to achieve good performance if the edge rewiring probability is distinct for different classes.

We apply the architecture of TAGCN to a COVID-19 case study. We introduce a novel approach for molecular property prediction by combining two existing GNN methods. Our model (D-MPNN+TAGCN) consistently outperforms the state-of-the-art baseline on five coronavirus datasets.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1: Introduction

Deep learning techniques such as convolutional neural networks (CNNs) have had major impact on fields like image processing, computer vision, and other Euclidean data domains, yet in many important domains data are irregular and non-Euclidean, requiring graphs or manifolds to be explicitly modeled. Examples of these contrasting data structures are shown in Fig. 1.1.

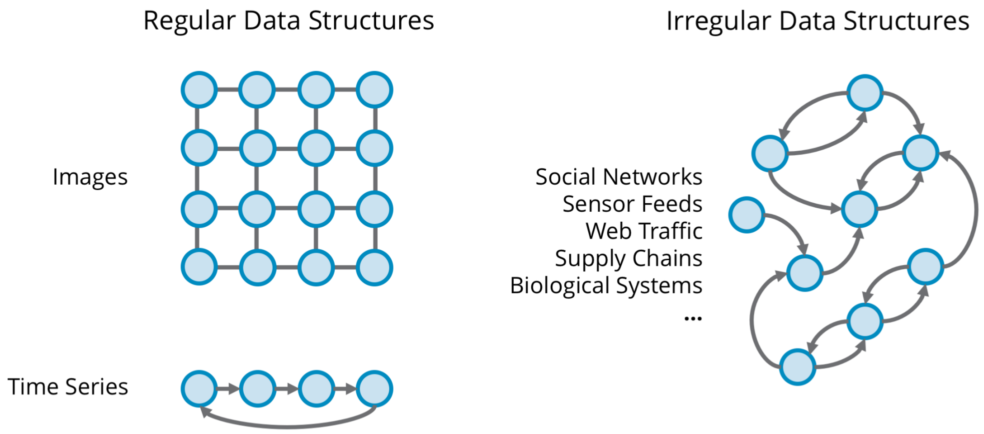


Figure 1.1: Problems in Euclidean data domains, like images and time series (left), have seen great advances through deep learning. Non-Euclidean data domains (right) require new deep learning techniques if similar advances are to be achieved.

Conventional deep learning approaches are often limited when data lack Euclidean structure to exploit. Fig. 1.2 shows the importance of successfully incorporating graph structure in the CNN architecture. The top plot illustrates that a CNN that takes advantage of the graph structure (termed here graph neural network or GNN) significantly outperforms a CNN that does not take advantage of the graph structure. The bottom plot assumes no graph, using only the MLPs to classify. We get similar results if instead of the Identity we use a random Erdős-Rènyi graph.

In this thesis, we explore the relationship between the data graph structure and the architecture of GNNs with respect to their performance in two classification tasks: node

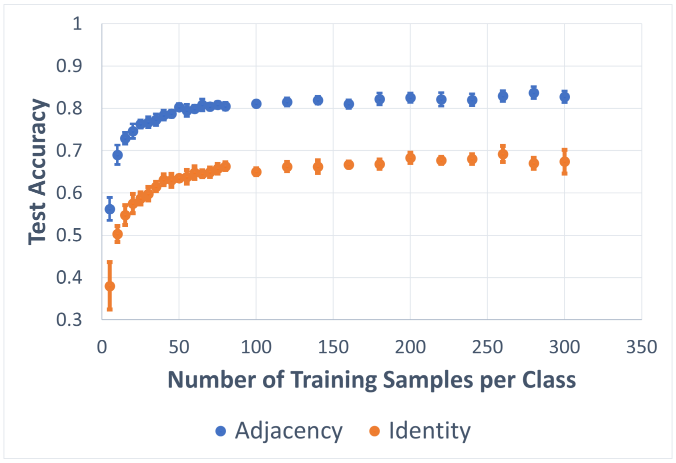


Figure 1.2: Comparison of GNN test accuracy on the CORA-ML citation network dataset using node features and 1) the dataset's citation network graph, 2) the identity matrix (training on features only)

and graph classification. In this chapter, we describe some related work in Section 1.1, including interpretability methods and automated machine learning for graphs. We then formally define the thesis statement in Section 1.2 and describe our contributions in Section 1.3. Finally, we present a short overview of the entire thesis in Section 1.4.

1.1 Related Work

In this section, we discuss related works in several fields.

1.1.1 Machine Learning on Graphs

Machine Learning (ML) is a field of inquiry devoted to developing algorithms that can leverage data to improve performance for some tasks [10]. It is an important subset of and pathway to artificial intelligence. The goal is for machines to learn how to solve problems without human intervention. In contrast, in traditional computer programming, algorithms are designed specifically, with every computation spelled out. ML algorithms hence allows for more flexibility in addressing more advanced problems.

The key feature of these ML algorithms is their capability to leverage large amount of data and extract useful information relevant to the task at hand [11]. ML algorithms build a model based on training data, and can make predictions on unknown data. For them to be successful, they need to operate within a carefully chosen framework, with right amount of complexity. More complex models will in general fit the training data better, but a model that is too complex will run the risk of over-fitting the random effects in training.

To extract structural information from graphs, machine learning approaches often rely on summary graph statistics (e.g., degrees or clustering coefficients), kernel functions, or carefully engineered features to measure local neighborhood structures. However, these approaches are limited because these hand-engineered features are inflexible (i.e., they cannot adapt during the learning process) and designing these features can be an expensive process. This then leads to requiring highly-efficient representation learning methods on non-Euclidean data

1.1.2 Deep Learning

The most popular method and subfield of machine learning is deep learning, which uses artificial neural networks (ANNs) [12]. ANNs originally attempt to mimic the human brain, inspired by the information processing and distributed communication nodes of neurons [13]. Alongside the growing success of these and other brain-inspired deep learning architectures, researchers have also used deep learning to understand and model brain activity [14].

Deep learning is called "deep" because of the multiple layers in the network. Popular neural network architectures in general have many layers. For example, ResNet, which won ImageNet 2015 challenge, has 152 layers [15]. The main advantage of deeper networks is that they can learn features at different levels of abstractions. Intuitively, when training ResNet on images, the initial layers detect lines and edges, the next level of layers capture polygons and shapes, the next level recognize even more complex shapes or collections of shapes like eyes or noses, and finally the highest level layers identify high-level features like faces. In contrast, wider layers are good at memorizing the features but not at generalizing (and hence leading to over-fitting).

1.1.3 Geometric Deep Learning

The extension of deep learning to non-Euclidean data is an area of research now called *geometric deep learning* [16]. The interest in this field has exploded in the past years, resulting in numerous successful attempts to apply these methods in a broad spectrum of problems from biochemistry and brain imaging to social networks and recommender systems (see Fig. 1.1). In many of these applications, there is the notion of relationships and interactions that are naturally occurring.

It is possible to represent these interactions as node features, where each feature represents a dimension [16]. This would allow the use of traditional deep learning like MLPs on these data. However, from an optimization perspective, higher dimensional data leads to an increase in the number of parameters and thus an increase in the dimension of the space where the optimization problem needs to be solved. From a statistical perspective, this is known as the curse of dimensionality. Initially, the machine learning algorithms improve up to a certain number of dimensions, but afterwards, the performance drops and levels off. An increasingly amount of data and computational cost are also necessary to solve the optimization problem. In a nutshell, the representation space imposed by MLPs become too complex.

Geometric deep learning addresses this issue by considering the data graph as an inductive bias instead of an additional feature. Essentially, it is a bias that can induce the model to learn certain patterns by changing the structure of the data. In many of the applications, the bias is relational. This restricts the representation space while retaining the flexibility and adaptability of the neural network.

1.1.4 Graph Neural Networks

One of the main algorithms in geometric deep learning is graph neural networks (GNNs). Just like how CNNs and recurrent neural networks are built for image-type and time-dependent data, respectively, GNNs are made for graph-based data. Many variants of GNNs are proposed to pass and aggregate information using the graph structure. These variants can be categorized into five groups: graph convolutional neural networks (GCNNs)[1], recurrent graph neural networks (RGNNs), graph pooling neural networks (GPNNs), graph autoencoders (GAEs), and Spatial-temporal Graph Neural Network (STGNNs). And there are a wide range of tasks these can be used for, including node classification, graph classification, representation learning, link prediction, anomaly detection, graph generation, community detection, graph embedding, and combinatorial optimization. This thesis focuses on node and graph classification tasks (see Chapters 4 and 5, respectively).

[1]Since the acronym "GCN" is used by [17] to means a specific model of graph convolutional neural networks

GNN has a strong theoretical foundation derived from graph signal processing. Graph signal processing (GSP), the generalization of traditional digital signal processing to graphs [18, 19, 20], can be used to extend CNNs to GNNs (see Section 2.3). We break down GNNs and analyze their components part by part in Chapter 3.

GNNs have several useful properties: generalization, expressivity, invariance, and transferability. Below, we summarize these properties. For more details, see [21, 22].

- **Generability**: GNNs are also among the most general class of neural network architectures, and CNNs (and hence MLPs) can be interpreted as special cases of GNNs. Reference [23] finds that GNNs have strong generalization guarantees if the largest absolute eigenvalue is independent of the graph size.

- **Expressivity**: For graph classification, GNNs can distinguish between very similar graphs. Reference [24] shows that GNNs are at most as powerful as the WL graph isomorphism test. In other words, if this test cannot distinguish between two non-identical graphs, then GNNs would not be able to either. For a more theoretical examination, see [25, 26].

- **Invariance**: There is no pre-fixed ordering of the nodes of a graph. We can permute the nodes of a graph by a permutation $\mathbf{P}$. This induces a permutation of the graph signal $\mathbf{P}\,\mathbf{X}$ and of the graph structure $\mathbf{P}\mathbf{A}\mathbf{P}^T$). GNNs are permutation equivariant functions $\mathbf{F}(\mathbf{X}, \mathbf{A})$ constructed by applying permutation invariant functions over local neighborhoods.

- **Transferability** GNNs have strong transferability, i.e., good test performance on graphs different than those are used in training. Reference [27] shows that the transference error decreases with graph size and there is a transferability-expressivity trade-off that can be alleviated by nonlinearities.

1.1.5 Automated Machine Learning for Graphs

The process of automating the tasks of applying machine learning is called automated machine learning (AutoML). The automation allows non-experts to quickly apply machine learning models to real-world problems and achieve state-of-the-art results. Architectures resulting from NAS have shown to outperform many manual architectures for object detection [28], image classification [29], hyperparameter optimization, [30], meta-learning [31], and neural architecture search [32].

We are interested here in hyperparamter optimization and neural architecture search for graph-based data. We can already apply AutoML to graphs, e.g., for hyperparameter optimization, via a grid search or a complete parameter sweep of the models. Our aim here is to gain a better understanding of how the graph structures can influence the performance of different hyperparamters and architectures, which can give insights into improving the performance and speed of AutoML for graphs. For example, for graph classification, graphs with high diameter may require a larger network (more graph convolutional layers).

1.1.6 Interpretability in Graph Neural Networks

Interpretability, or understanding the cause of the model's performance, is an important focus of AI. The higher the interpretability, the easier it is to understand the decisions or output accuracy based on the inputs. It is especially important in high-stakes domains like healthcare and and criminal justice. Many state-of-the-art ML models used today are black boxes that do not explain their predictions. Despite having the best prediction accuracy, their lack of transparency had severe consequences. For example, ML models have found that highly polluted air in Sacramento was safe to breath [33] and denied parole based on race [34].

There are 2 main classes of methods for interpretability for GNNs: instance-level methods and model-level methods. Instance-level methods are more popular and identify the components crucial for the model's prediction. There are four main subclasses of instance-level methods and one model-level method:

- **Gradients- or features-based methods** use the backpropagated gradients or feature map. For example, class activation mapping [35] takes the final node embedding to create a weighted activation map for the input that highlights the most crucial components.

- **Perturbation-based methods** perturb the input and monitor the change in output. For example, GNNExplainer [36] uses mutual information between GNN's prediction and distribution of possible subgraph structures to identify the important subgraph and node features.

- **Decomposition methods** decompose the prediction scores (e.g., softmax output) and propagate the scores back to the input layer. An example is GNN-s layer-wise relevance propagation (GNN-LRP) [37] that decomposes the output and follows a backpropagation procedure to uncover the graph walks that are important for the prediction.

- **Surrogate methods** first sample a dataset using local neighborhood sampling, node feature perturbation, etc. Then fit different surrogate models (such as decision tree). An example is RelEx [38], which combines surrogate methods and perturbation methods. It considers N-hop neighboring nodes for random sampling and employs a GCN model as the surrogate model.
- **Model-level methods** do not use any specific input configuration. The only approach, XGNN from [39] trains a graph generator to maximize a target graph prediction. The generated graph patterns serve as explanations for the prediction.

For a complete review of the taxonomy, see [40]. Our approach is closest to the perturbation-based methods. We are interested in a high-level interpretability using a data-based approach by generating synthetic data, and measuring the effects on the performance and the hyperparameters.

1.2 Thesis Statement

This thesis explores the neural network architectural considerations given the graph topology underlying the data. We focus on two applications: node classification and graph classification. We measure the performance by test accuracy. For node classification, we also use training convergence as a performance metric. For each application, We consider common real-world benchmarks, and three models of random networks. We also look into one real-life application: COVID-19 drug discovery.

Thesis Statement: With appropriate architectures and algorithm design, geometric deep learning methods are capable of providing general solutions to graph-based data. To achieve high performance, it is important to consider the data graph structure when designing the neural network architecture.

The guidelines and insight developed in this thesis will enable researchers to take better advantage of the power of geometric deep learning to solve problems in different fields where the underlying data structure is a graph. All the codes and data developed when completing this thesis are publicly available.

1.3 Contributions

This thesis aims to be a step towards understanding how to build better performing graph neural network architectures. Towards this end, our major contributions in this thesis are as follows:

- We present a mathematical framework for graph convolutional neural networks based on graph signal processing (GSP). We discuss how GSP can be used with CNNs, leading to GNNs. GSP provides a theoretical framework and intuition about these operations.
- We review 3 main components of the GNN architecture: graph convolutional layers, graph pooling layer, and graph aggregation layer. This thesis focuses on the graph convolutional layer, and, to a lesser degree, on the graph aggregation layer.
- **Node classification**. We study how the data graph structure affects the GNN architecture for the node classification task. In particular:
  - ***Real datasets***. We analyze the effects of graph structure on test accuracy for seven real datasets. We learn that:
    - Not many layers are needed to achieve optimal performance. TAGCN requires fewer number of layers than GCN, even with low degree of the polynomial filters. Unlike graph classification (studied in Chapter 5), the graph signal is necessary and important to achieve good classification results.
    - For datasets with higher difference in % of intraclass and interclass edges, the effect of over-smoothing due to number of layers is smaller.
    - Over-smoothing occurs at a slower rate as we increase the degree of polynomial filters for datasets with low average degree.
  - ***Three random models***. We analyze the effects on test accuracy of different TAGCN and GCN designs for 3 graph generation models (Erdős-Rényi, smallworld, and preferential attachment). Our main conclusions include:
    - Erdős-Rényi and preferential attachment models have similar test accuracy curves for both GCN and TAGCN with respect to the number of layers and the degree of the polynomial filters. For smallworld model,

TAGCN's filters play an important role in achieving the optimal accuracy and accelerating the effect of over-smoothing.

    * High intraclass edge ratio not only leads GNN to achieve higher accuracy, but also leads them having greater resistance to over-smoothing with respect to both number of layers and degree of polynomial filters. We observe that polynomial filters have higher optimal degrees for lower layers and lower optimal degrees for higher layers.
    * Changing the graph signals by increasing the interclass difference between the means of the graph signal improves the accuracy, while not affecting the optimal number of layers and the degree of the polynomial filters.
    * Increasing the number of nodes for the preferential attachment synthetic network only lowers the variance of the test accuracy for both architectures if all other graph characteristics remain the same.

- **Graph classification**. We study how graph structure affects the performance of GNN architecture for the graph classification task:
  - ***Real datasets***. We analyze the effects of graph structure on test accuracy for 11 real datasets. We learn that:
    * Datasets with high average degree and low average diameter require less number of layers to achieve optimal performance. TAGCN tends to perform better with higher degrees of the polynomial filters at low number of layers on datasets with low average degree.
    * We relate simple network metrics and signal statistics to the performance of these models. We show for biological network datasets, classifiers on number of edges or number of nodes can lead to better or similar performance as graph neural networks. For social network datasets, signal statistics can perform well. Based on these observations, we were able to apply simple modifications to GCN and TAGCN to improve their performance (sumpool and degree-aware TAGCN).
  - ***Three random models***. We analyze the effects on test accuracy of different TAGCN and GCN design for 3 graph generation models (Erdős-Rényi, smallworld, and preferential attachment). Our main conclusions include:
    * Erdős-Rényi and preferential attachment models have similar test accuracy curves for both GCN and TAGCN. For smallworld model, more than

    1 layer is needed to achieve good performance if the edge rewiring probability is distinct for different classes.

  * Simple network metrics that capture different interclass graph characteristics can perform better than GNNs (e.g., clustering coefficient performs better than TAGCN for capturing the smallworld's edge rewiring probability).
  * Just like with real datasets, we also show that a simple modification (degree-aware TAGCN) can greatly improve the performance in capturing interclass difference in degree. However, another popular variant (graph attention network) does not perform better than TAGCN on capturing these graph characteristics.
  * We show that while TAGCN can perform well when graph signal and graph structure have an AND relationship, it fails when they have an XOR relationship.

- **Theoretical analysis: Convergence**. We prove that training loss for linearized TAGCN converge to a global minimum at a linear rate is guaranteed under mild assumptions. We show how the training can be accelerated with higher order polynomial filters and more depth. Our experiments confirm that our results on real datasets are aligned with our theoretical results for linearized TAGCNs, with and without non-linearity.

- **Covid 19 study**. We apply the architecture of TAGCN to a COVID-19 case study. We introduce a novel approach for molecular property prediction by combining two existing GNN methods. Our model (D-MPNN+TAGCN) consistently outperforms state-of-the-art baseline (D-MPNN) on five coronavirus datasets.

1.4 Thesis Overview

We first present preliminaries on graph theory, deep learning, and graph signal processing in Chapter 2. We then study in Chapter 3 the GNN architecture by considering its main blocks, i.e., convolutional layer, pooling layer, and aggregation layer. Next, in Chapter 4, we apply GNNs to the task of node classification and analyze the performance with respect to the graph structure and GNN architecture on real and synthetic datasets. We consider the analysis for the graph classification task in Chapter 5 where the data graph structure plays a more important role. In Chapter 6, we prove the training convergence for our main architecture for node classification and validate the theory experimentally on real datasets. Finally, we apply GNN to a COVID-19 case study in Chapter 7 and conclude in Chapter 8.

Chapter 2: Preliminaries

In this chapter, we provide background information on graph theory, deep learning, and graph signal processing that are necessary to understand the data graph structure and graph neural networks. We start with a brief review of graph theory and network algorithms (Section 2.1). We then discuss the architecture of CNNs, which is analogous to GNNs in certain aspects (Section 2.2). Then we end with a primer on graph signal processing that provides a theoretical framework to design and evaluate GNNs (Section 2.3).

2.1 Preliminaries on Graph Theory

In this section, we review some basic definitions and notations from spectral graph theory, with a focus on how to extend many of the important mathematical ideas and intuitions from classical Fourier Analysis to the graph setting. This is foundational to understand spectral graph filtering and convolution, concepts that are important with graph neural networks.

2.1.1 Definition and Notations

Graph structure, defined by its adjacency matrix, may differ from problem domain to problem domain to a far greater extent than image structure in different computer vision problems. We are interested in analyzing signals defined on a connected graph $\mathcal{G} = (V, \mathbf{X}, \mathcal{E}, \mathbf{A})$, which consists of a set of nodes V with $|V| = N$, signal $\mathbf{X} \in \mathbb{R}^{N \times C}$ on the nodes (where C is the number of features on each node), a set of edges $\mathcal{E}$, and an adjacency matrix $\mathbf{A}$. $\mathbf{A}_{ij} = 0$ unless there is an edge $e = (i, j)$ connecting node i to node j. The graph is undirected if $\forall (i, j) \in \mathcal{E} \mid \mathbf{A}_{ij} = \mathbf{A}_{ji}$. For many datasets, the adjacency matrix is unweighted, i.e., the entries in $\mathbf{A}$ are either zero or one.

In some applications, the graph structure is naturally defined. When this is not the case, there are many alternative ways to define a graph that underlies the data. For example, there are various distances (Euclidean, Manhttan, Minkowski). One can define a graph for any data using the distances between each pair of nodes and thresholding. Alternatively, one can use a k-nearest neighbors graph. Other optimization methods include causal graph processes [41] and principle network analysis [42] for time-varying graphs.

2.1.2 Graph Laplacian

Definition 1 (graph Laplacian)**.** The graph Laplacian is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix of $\mathcal{G}$, defined as the diagonal matrix $(\mathbf{D})_{ii} = \sum_i (\mathbf{A})_{ij} = \sum_j (\mathbf{A})_{ij}$.

Graph Laplacians apply only to undirected graphs, so they are symmetric and positive semidefinite, with real non-negative eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. Eigenvalues of 0s appear with multiplicity equal to the number of connected components of the graph. The eigenvectors are orthogonal but do not necessarily form a unique set. For directed graphs, either the indegree or the outdegree may be used. Graph Laplacians have been widely studied within the field of spectral graph theory [20].

Definition 2 (normalized graph Laplacian)**.** For undirected graphs, we can use the normalized graph Laplacian, i.e.,

$$\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5} = \mathbf{D}^{-0.5}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-0.5} = \mathbf{D}^{-0.5}(\mathbf{L})\mathbf{D}^{-0.5}. \tag{2.1}$$

The eigenvalues $\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_n$ of a normalized graph Laplacian for connected, undirected graph $\mathcal{G}$ satisfy:

$$0 \leq \tilde{\lambda}_1 \leq \tilde{\lambda}_2 \leq \tilde{\lambda}_n \leq 2$$

with $\tilde{\lambda}_n = 2$ if and only if $\mathcal{G}$ is bipartite. Both the regular graph Laplacian and normalized graph Laplacian can be used as a filtering basis for the graph Fourier transform in GSP and the convolution in graph neural networks when the data graph is undirected.

2.1.3 Graph Statistics and Network Metrics

Graph structure and signal (or feature) have a significant effect on the performance of GNNs. For example, if the graph feature is the same across all the nodes, node classification setting would perform as well as random guessing. To quantify the graph signal, we consider the signal mean and variance for graph classification, and signal similarity for node classification. In general, for node classification, GNNs perform better if the signals between nodes of the same class are similar and distinct for different classes. Likewise, for graph classification, higher difference in signal means leads to better classification.

To quantify the graph structure, we consider several graph statistics and network metrics. We consider number of nodes, number of edges, average degree, clustering coefficient, and a combination of these. We also consider other network metrics such as degree centrality, number of triangles, etc. For a full list of the network metrics, see [43]. In general, for node classification, if the clustering coefficient for nodes of similar classes are high, GNNs perform better. For graph classification, default GNNs are not equipped to detect differences in number of nodes and edges.

2.2 Preliminaries on Deep Learning

CNNs are particularly effective among deep learning models, in part because of the shift-invariance of convolutional filters. A simple CNN architecture is shown in Fig. 2.1, and discussed in further detail in section 3.2. From the figure, the basic building blocks are convolution, pooling, activation and fully connected layers. In this section, we discuss each layer in detail, with a particular focus on convolution, pooling, and aggregation layers. We will show in Chapter 3 how they are different for GNNs.

2.2.1 Convolutional Layer

Convolutional layers are fundamental building blocks of CNNs. They are a linear operation that filters an input signal. It involves the multiplication of a set of weights with the input. The resulting dot product is passed to the next layer. It exploits spatially local correlation by learning through each filter a local connectivity pattern between the input data. For example, a basic operation of a CNN convolutional kernel is shown in detail in Fig. 2.1, where a 3×3 filter or kernel (red window) slides over the input ($\mathbf{X}$) to generate the output. Initial layers have a small receptive field and tend to learn the edges and lines of the image. Each subsequent layer has larger receptive fields (since they take in as inputs the previous filter outputs that learned local structure), till the last layers for which the receptive field covers the entire image and can classify it as a whole.

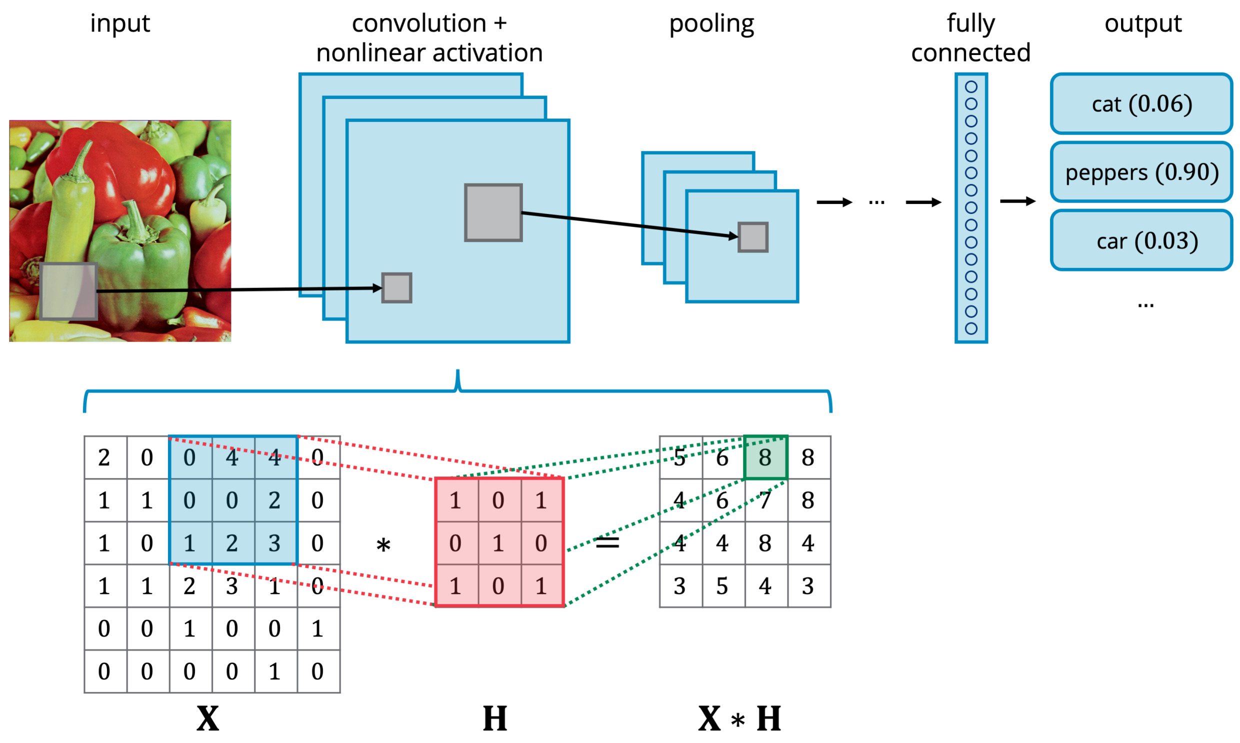


Figure 2.1: Basic CNN architecture and kernel. A typical CNN consists of several component types (e.g., several convolutional + nonlinear activation layers, interleaved with pooling layers, followed by fully connected layers before prediction). The convolutional kernel sums the point-wise multiplications of a subset of the signal **X** with a learned filter **H**. This operation is repeated as the filter "slides" across the input. (Note that what is termed convolution by the deep learning community is actually cross-correlation.)

Another powerful property of convolutional layers is parameter sharing. It relies on the fact that if a patch is useful at some spatial location, it can also be useful at other locations. For example, if a filter can detect the left eye, it can also be used to detect the right eye. Essentially, parameter sharing allows for translational equivariance in CNNs, which means we can move the subject in the image vertically and/or horizontally without hurting performance. Parameter sharing also allows for a small memory footprint and computational cost. In constructing a GNN, these properties should ideally also be preserved.

2.2.2 Pooling Layer

Pooling can be understood as a nonlinear downsampling operation by summarizing the presence of features in patches. Just like the convolutional layer, the pooling layer uses a moving filter. Most common pooling methods are average pooling and max pooling, which take the mean and the max of a filter, respectively.

The pooling layer reduces the size of the image while preserving the most important features. The intuition is that the exact location is not needed for classification. Hence, by reducing the spatial size of the representation, the number of parameters is decreased, preventing overfitting. The computational efforts in the network are also reduced. Generally, pooling layers are periodically inserted between successive convolutional layers, providing another form of translation invariance.

2.2.3 Activation Layer

Activation functions introduce nonlinearity. Without them, the model performs only a linear combination of the input data, underperforming when the classes are not linearly separable (e.g., classifying a circle from its background). The most popular activation function is Rectifying Linear Unit (ReLU):

$$\mathrm{ReLU}(x) = \max(0, x). \tag{2.2}$$

We generally use ReLU, and analyze training convergence with and without it.

2.2.4 Fully Connected Layer

Finally, right before prediction and the final activation layer, inputs are flattened and passed to fully connected layers/ multilayer perceptron (MLP). These layers are used to separate the features captured by the previous layers.

2.3 Preliminaries on Graph Signal Processing

Graph signal processing (GSP) extends traditional signal processing operations such as shifting, Fourier transforms, convolution, and sampling to graphs [18, 20]. GSP defines these operations for graph data and provides a theoretical framework to design and evaluate GNNs. In this section, we give a primer on GSP theory, which is useful to study the GNN architecture and analysis in the next sections.

GSP has been developed to process, from first principles, graph-structured data [18, 20]. GSP can be thought of as a generalization of classical signal processing: whereas the structure of classical signals is implicit, the structure of non-Euclidean data must be explicitly represented, which GSP does with the pairwise relationships captured by graphs. Because GSP is a generalization of classical signal processing, as developed in [18], GSP reduces to DSP in the special case of a uniform graph (e.g., the pixels of an image [44] or indices of a time series).

GSP can perhaps best be intuited by beginning with a generalization of the shift operation from DSP. (A more thorough introduction to GSP can be found in [18, 20].) A finite support (or periodic) discrete-time signal $x[n]$ with period N can be represented by a vector $x = [x_1, x_2, \dots, x_{N-1}, x_N]^T$. In this representation, to filter $x[n]$ by some finite impulse response (FIR) filter g, we represent g as a matrix $\mathbf{G}$ and simply perform a matrix multiplication $x' = \mathbf{G}x$.

Under appropriate conditions[1], any linear, shift-invariant filter, $\mathbf{G}$, can be expressed as a polynomial of shifts; the shift operator plays a crucial role in DSP. For the periodic time model and DSP, we can choose the circular shift:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix}. \tag{2.3}$$

For this shift operator $\mathbf{A}$, shift operation on x is:

$$\mathbf{A}x = \mathbf{A}[x_1, x_2, \dots, x_{N-1}, x_N]^T = [x_N, x_1, \dots, x_{N-2}, x_{N-1}]^T. \tag{2.4}$$

We can interpret this DSP operation as a graph operation by recasting $\mathbf{A}$ as an adjacency matrix—a matrix representation of a graph. More precisely, $\mathbf{A}$ corresponds to the adjacency matrix of a directed ring graph $\mathcal{G} = (V, \mathcal{E})$, where V is a set of N vertices and $\mathcal{E}$ the set of directed edges connecting each vertex to its next neighbor (with the final vertex connecting back to the first). The N elements of x are indexed by the N vertices of $\mathcal{V}$, and each entry $\mathbf{A}_{ij}$ is the weight w_{ij} of the edge connecting vertex i to j.

This dual role played by the matrix $\mathbf{A}$ is at the heart of constructing a linear, shift invariant GSP, as it allows us to extend the concept of shift to any arbitrary graph [18]. The graph interpretation of equation (2.4) is shown in Fig. 2.2.

We note here that graph shift operators other than $\mathbf{A}$ have been proposed [18, 20], such as the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix of $\mathcal{G}$, defined as

[1]The characteristic and minimal polynomials are the same. We assume the sufficient condition that the eigenvalues of $\mathbf{A}$ are distinct.

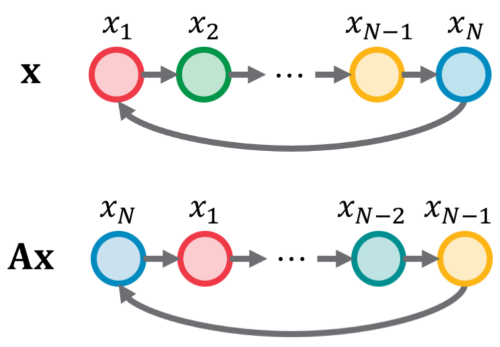


Figure 2.2: A graph signal x residing on a ring graph (top), equivalent to a time series, and its shifted version $\mathbf{A}x$ (bottom). The color of each node denotes the data that is indexed by that node. The action of the shift is represented by the shift (by one hop) of the colored nodes to the right on the right graph.

the diagonal matrix $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. Graph Laplacians apply only to undirected graphs, but they are symmetric and positive semidefinite. They have been widely studied within the field of spectral graph theory [20].

2.3.1 Convolutional Layer

Building naturally upon the graph shift operator $\mathbf{A}$, graph convolution is the matrix vector multiplication:

$$\mathbf{G}\,x = \mathrm{g}(\mathbf{A})x = \sum_{k=0}^{K} \alpha_k \mathbf{A}^k x \tag{2.5}$$

where $\mathrm{g}(\mathbf{A})$ is the (polynomial) filter of degree K, x is the graph signal, and α_k the k-th filter coefficient. In practice, $\mathbf{A}$ is often normalized in some manner to ensure numerical stability. For example, normalizing the shift by $|\lambda_{\max}|$ where $\lambda_{\max}$ is the eigenvalue of $\mathbf{A}$ with greatest magnitude, or, with an undirected graph, using $\bar{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ in place of $\mathbf{A}$, where $\mathbf{D}$ is the degree matrix of $\mathbf{A}$. These guarantee that the (non-maximal) eigenvalues of the adjacency matrix are inside the unit circle, thereby making $\mathbf{G}$ computationally stable.

The linear, shift invariant properties of $\mathbf{G}$ carry important implications. If we reorder the nodes of $\mathbf{A}$, then we get an adjacency matrix $\mathbf{A}_{\text{new}} = \mathbf{P}\mathbf{A}\mathbf{P}^T$, where $\mathbf{P}$ is a permutation matrix. This also reorders the graph signal x to become $x_{\text{new}} = \mathbf{P}x$. Using (2.5),

$$\mathbf{G}_{\text{new}}x_{\text{new}} = \mathrm{g}(\mathbf{A}_{\text{new}})x_{\text{new}} = \sum_{k=0}^{K} \alpha_k \mathbf{A}_{\text{new}}^k \mathbf{P}x = \sum_{k=0}^{K} \alpha_k \mathbf{P}\mathbf{A}^k\mathbf{P}^T\mathbf{P}x = \mathbf{P}\sum_{k=0}^{K} \alpha_k \mathbf{A}^k x = \mathbf{P}\left(\mathbf{G}x\right). \tag{2.6}$$

Thus, if we reorder the nodes by a permutation matrix $\mathbf{P}$ and convolve, we obtain the original result reordered by the same permutation. This is a desirable property for learning architectures because reordering the nodes in $\mathbf{A}$ should not affect the inference result.

2.3.2 Spectral Analysis: Graph Fourier Transform

GSP provides a mathematical framework that unifies the vertex and spectral domains of a graph, much as classical signal processing connects the time and frequency domains of a time-series.

In DSP, the discrete Fourier transform (DFT) is equivalent to the eigendecomposition of the directed ring graph adjacency matrix $\mathbf{A}$ from (2.3):

$$\mathbf{A} = \mathrm{DFT}^H \mathbf{\Lambda}\, \mathrm{DFT}. \tag{2.7}$$

Similarly, in GSP, we define the graph Fourier transform (GFT) by the eigendecomposition of an arbitrary adjacency matrix $\mathbf{A}$.

$$\mathbf{A} = \mathrm{GFT}^{-1} \mathbf{\Lambda}\, \mathrm{GFT}. \tag{2.8}$$

Given x, the graph spectral representation of the graph signal is $\widehat{x} = \mathrm{GFT}x$.

2.3.3 Pooling Layer

GSP can also be used to extend sampling theory. Assume a $N \times 1$ signal x. We can sample x by choosing $K < N$ values to keep and zeroing out the rest. This operation can be represented as $x \odot \delta$, where

$$\delta_i = \begin{cases} 1, & \text{if } x_i \text{ is chosen} \\ 0, & \text{if } x_i \text{ is not chosen} \end{cases} \tag{2.9}$$

and $\odot$ is the Hadamard or element wise product of two vectors, $|\delta|_0 = K$, where $|\cdot|_0$ stands for the zero pseudonorm given by the number of nonzero components of δ.

If $\widehat{x}$ is bandlimited with bandlimit K, i.e., $\widehat{x}$ contains K nonzero values, then we can choose a sampling set, represented by its characteristic function δ, such that x is perfectly recoverable from $x \odot \delta$. Reference [45] provides two criteria for choosing δ such that x is perfectly recoverable from $x \odot \delta$, one in the vertex domain and one in the spectral domain. These criteria both involve choosing linearly independent rows and columns from parts of the GFT (for vertex) or GFT^{-1} (for spectral) matrices.

2.4 Conclusion

To summarize, we provide in this chapter background information for the thesis in 3 parts. First, we review some basic definitions and notations from spectral graph theory, and present some simple graph statistics and network metrics that will be considered in future chapters. Then we present the building blocks of convolutional neural networks, which will be extended to graph neural networks. Finally, we give a primer on graph signal processing, which serves as the theoretical foundation for GNNs.

Chapter 3: Graph Neural Networks

In this chapter, we study the graph neural network (GNN) architecture by considering its main blocks. We discuss 2 flavors of the graph convolutional layer: spectral and spatial (Section 3.2). Then we consider pooling (Section 3.3) and aggregation (Section 3.4) in detail, highlighting their design intuitions. Content of this chapter is published in [2].

3.1 Introduction

The CNN architecture has been successfully applied to many classification tasks like image classification and speech recognition. Fig. 2.1 shows the basic architecture of a CNN. The deep model transforms an input through a sequence of hidden layers. Each early-stage hidden layer consists of a set of learned filters, like convolution followed by a nonlinear activation function, or pooling; a filter outputs a feature map that serves as the input to the subsequent layer. Later hidden layers are typically fully connected and, finally, a prediction is made at the output layer, using a loss function like cross-entropy loss. Numerous variants and their improvements have been detailed across the literature.

A GNN architecture can likewise be constructed from convolutional, pooling, activation and fully connected layers. Convolutional and pooling layers can be constructed anew to incorporate graph structure using GSP theory. We describe these layers in this section and explore GNNs for node and graph classification tasks.

3.2 Graph Convolutional Layer

CNNs, like that shown in Fig. 2.1 in section 2.2, have proven so effective in domains like computer vision because the convolutional kernel has several powerful properties: 1) it has a fixed number of parameters, allowing for a comparatively small memory footprint and computational cost; 2) it operates locally, meaning higher-level, global features can be composed of lower-level, local features; and 3) it is shift invariant. In constructing a GNN, such properties should be kept. The basic operation of a CNN convolutional kernel can be seen in Fig. 3.1.

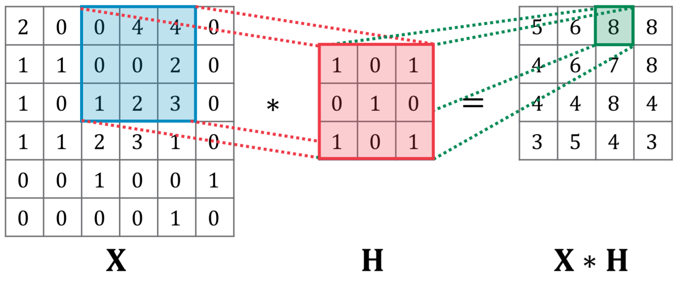


Figure 3.1: The convolutional kernel sums the point-wise multiplications of a subset of the signal **X** with a learned filter **H**. This operation is repeated as the filter "slides" across the input. (Note that what is termed convolution by the deep learning community is actually cross-correlation.)

Broadly speaking, two approaches to GNNs have been pursued, the spectral domain approach and the vertex domain approach.

Graph convolution was first generalized from CNNs to graph-structured data in the spectral domain using the graph Laplacian. The spectral approach takes the graph signal x, multiplies by the GFT to get $\hat{x}$, convolves in the spectral domain, and then multiplies by GFT^{-1} to return to the vertex domain [46]. To avoid the expensive eigendecomposition of $\mathbf{A}$ to find the GFT matrix, [47] used Chebyshev polynomials to approximate the GFT.

The vertex approach defines convolution in the vertex domain, as given by (2.5). These approaches are respectively typified by graph convolutional networks (GCNs) [17] and topology adaptive graph convolutional network (TAGCNs) [48], which we consider below. Other implementations, like GraphSAGE [49] and graph attention networks (GATs) [50], are deep learning approaches defined in the vertex domain, but their kernels do not meet the definition of convolution in GSP.

Given a graph signal $\mathbf{X}^{(0)} \in \mathbb{R}^{N\times C}$ (where N is the number of nodes, and C the number of signal dimensions, or channels) and a graph structure $\mathbf{A} \in \mathbb{R}^{N\times N}$, a GCN graph convolutional layer in general form is [17]:

$$\mathbf{X}^{(\ell+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}^{(\ell)}\mathbf{W}^{(\ell)}\right) \tag{3.1}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$, $\mathbf{W}^{(\ell)} \in \mathbb{R}^{C \times F}$ the trainable weight matrix, σ the nonlinear activation function, and F the number of output features. The layer number is given by ℓ, with $\ell = 0$ for the input layer. Adding the identity matrix to $\mathbf{A}$ and then normalizing helps address numerical instabilities and vanishing gradients.

The TAGCN implementation of graph convolution [48] treats the polynomial filter coefficients as learnable weights. We can write the general form of the TAGCN graph convolutional layer as:

$$\mathbf{X}^{(\ell+1)} = \sigma\left(\sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}^{(\ell)} \mathbf{W}_k^{(\ell)}\right). \tag{3.2}$$

Fig. 3.2 shows a TAGCN Polynomial filter of degree 2 for the blue node. Signals at the red nodes are propagated to the blue one and aggregated (summed), i.e.,

$$\mathbf{A}^2\mathbf{X}^{(\ell)}\mathbf{W}_2^{(\ell)} + \mathbf{A}\mathbf{X}^{(\ell)}\mathbf{W}_1^{(\ell)} + \mathbf{I}_N\mathbf{X}^{(\ell)}\mathbf{W}_0^{(\ell)}.$$

As mentioned before in section 2.3.1, we can use the normalized versions of $\tilde{\mathbf{A}}$ and $\mathbf{A}$ in (3.1), (3.2).

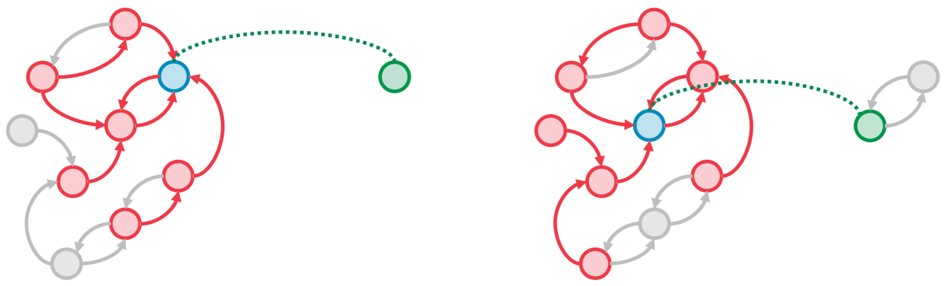


Figure 3.2: Example of a graph convolutional filter of degree 2. (Left) The filter, centered on the blue vertex, aggregates neighborhood information by multiplying the graph signal with a polynomial of shifts. For a degree of 2, the graph filter information from first- and second-order neighbors is used to compute the output value. The polynomial coefficients are learned, similar to the filter coefficients of a CNN. (Right) Like a CNN, the filter “slides” across the graph from vertex to vertex, and the output is fed to a nonlinear activation function.

3.2.1 Experiments

To get some intuitions about the performance of GCN and TAGCN, we present some experimental results. We compare GCNs [17] and TAGCNs [48] for node classification and graph classification on popular benchmark datasets. See Sections 4.3.2 and 5.3.2 for the node and graph classification experimental setup, respectively. Table 4.1 lists node classification datasets and their data statistics. Tables 5.2 and 5.3 list the biological and social graph classification datasets and their data statistics.

Table 3.1: Comparison of GNN Variants for Node Classification

	CORA-ML	CiteSeer	PubMed
GCN	78.8 ± 1.7	67.9 ± 1.9	77.3 ± 2.4
TAGCN	$\mathbf{80.6 \pm 1.6}$	$\mathbf{68.5 \pm 1.8}$	$\mathbf{77.6 \pm 2.5}$

Table 3.2: Comparison of GNN Variants for Graph Classification

	MUTAG	PROTEINS	IMDB-B	REDDIT-B	COLLAB
GCN	74.0 ± 7.3	$\mathbf{72.8 \pm 1.7}$	72.8 ± 2.3	88.2 ± 1.8	79.5 ± 2.0
TAGCN	$\mathbf{75.1 \pm 8.2}$	72.4 ± 2.9	$\mathbf{73.3 \pm 5.3}$	$\mathbf{91.6 \pm 2.6}$	$\mathbf{81.0 \pm 1.1}$

Tables 3.1 and 3.2 show the performance of several variants of GNNs for node and graph classification, respectively. In general, TAGCN performs better than GCN for both classification tasks. However, a GCN filtering operation has complexity $\mathcal{O}(|\mathcal{E}|FC)$, whereas the same operation for TAGCN has complexity $\mathcal{O}(|\mathcal{E}|KFC)$ where K,F,C are the degree of the polynomial filter, the number of output features, and the number of channels, respectively. Due to the increased complexity, TAGCN has higher variance, especially for denser graph structures for graph classification task (since for node classification, there is just one graph). Empirically, TAGCN achieves higher classification accuracy as graphs become less sparse, i.e., as average degree increases.

3.2.2 Convolution and GSP

In this section, we relate the graph convolutional layer in GCN [17] and TAGCN [48] to graph signal processing. From (2.5), each graph convolutional layer can be interpreted in GSP as:

$$x' = \sigma\left(\mathrm{g}(\mathbf{A})x\right) \tag{3.3}$$

where x is the graph signal (input to the layer), x' the output of the layer, $\mathrm{g}(\mathbf{A})$ a polynomial of degree K and σ the nonlinear activation. By (2.8), one convolutional layer is

$$\mathrm{g}(\mathbf{A})x = \mathrm{GFT}^{-1}\mathrm{g}(\mathbf{\Lambda})\mathrm{GFT}x = \mathrm{GFT}^{-1}\mathrm{g}(\mathbf{\Lambda})\widehat{x} \xrightarrow{\mathrm{GFT}} \mathrm{g}(\mathbf{\Lambda})\widehat{x}. \tag{3.4}$$

If there are L convolutional layers in the architecture, this can be interpreted as a cascade of L polynomial filters:

$$g_L(\mathbf{A})g_{L-1}(\mathbf{A})\ldots g_1(\mathbf{A})x. \tag{3.5}$$

Using (2.8) yields

$$\begin{aligned} g_L(\mathbf{A})g_{L-1}(\mathbf{A})\dots g_1(\mathbf{A})x &= \mathrm{GFT}^{-1}\mathrm{g}_L(\mathbf{\Lambda})\mathrm{GFT}\dots\mathrm{GFT}^{-1}\mathrm{g}_2(\mathbf{\Lambda})\mathrm{GFT}\ \mathrm{GFT}^{-1}\mathrm{g}_1(\mathbf{\Lambda})\mathrm{GFT}x \\ &= \mathrm{GFT}^{-1}\mathrm{g}_L(\mathbf{\Lambda})\mathrm{GFT}\dots\mathrm{GFT}^{-1}\mathrm{g}_2(\mathbf{\Lambda})\mathrm{GFT}\ \mathrm{GFT}^{-1}\mathrm{g}_1(\mathbf{\Lambda})\widehat{x} \\ &\xrightarrow{\mathrm{GFT}} \mathrm{g}_L(\mathbf{\Lambda})\dots\mathrm{g}_2(\mathbf{\Lambda})\mathrm{g}_1(\mathbf{\Lambda})\widehat{x} \end{aligned} \tag{3.6}$$

where Λ is the diagonal matrix of eigenvalues. Equation (3.5) is filtering in the vertex domain by cascading product of $\mathbf{A}$ while (3.6) is filtering in the spectral domain by first taking the GFT and GFT^{-1}. In the time domain, the fast Fourier transform (FFT) reduces the computation dramatically. Going through the spectral domain requires taking the GFT, which may be expensive. GFT produces matrices that are dense when compared to graphs in the vertex domain (which are generally sparse), hence requiring much larger computational resources

Reference [46] suggested using a filter in the spectral domain. Based on this, we also consider a spectral domain approach to GCN and TAGCN based on GSP, dual to the vertex domain approach described above.

Since $\mathbf{M}$, the spectral domain shift, is the graph spectral dual of $\mathbf{A}$, we define the graph convolutional layer in the spectral domain as:

$$\mathrm{g}(\mathbf{M})\widehat{x} = \mathrm{GFT}\ \mathrm{g}(\mathbf{\Lambda}^*)\mathrm{GFT}^{-1}\widehat{x} = \mathrm{GFT}\ \mathrm{g}(\mathbf{\Lambda}^*)x \xrightarrow{\mathrm{GFT}^{-1}} \mathrm{g}(\mathbf{\Lambda}^*)x. \tag{3.7}$$

A cascade of L polynomial filters in the spectral domain is

$$\begin{aligned} g_L(\mathbf{M})g_{L-1}(\mathbf{M})\dots g_1(\mathbf{M})\widehat{x} &= \mathrm{GFT}\mathrm{g}_L(\mathbf{\Lambda}^*)\mathrm{GFT}^{-1}\dots\mathrm{GFT}\mathrm{g}_2(\mathbf{\Lambda}^*)\mathrm{GFT}^{-1}\mathrm{GFT}\mathrm{g}_1(\mathbf{\Lambda}^*)\mathrm{GFT}^{-1}\widehat{x} \\ &= \mathrm{GFT}\mathrm{g}_L(\mathbf{\Lambda}^*)\mathrm{GFT}^{-1}\dots\mathrm{GFT}\mathrm{g}_2(\mathbf{\Lambda}^*)\mathrm{GFT}^{-1}\mathrm{GFT}\mathrm{g}_1(\mathbf{\Lambda}^*)x \\ &\xrightarrow{\mathrm{GFT}^{-1}} \mathrm{g}_L(\mathbf{\Lambda}^*)\dots\mathrm{g}_2(\mathbf{\Lambda}^*)\mathrm{g}_1(\mathbf{\Lambda}^*)x \end{aligned} \tag{3.8}$$

Equations (3.6) and (3.8) provide a GSP interpretation of graph convolutional layers for both the vertex and spectral domains respectively. We examine many possible convolutional layers with a nonlinear activation in both the vertex and spectral domain by placing the nonlinear activation σ in different locations in (3.6) and (3.8). Table 3.3 contains the accuracy of these convolution kernels for different datasets (see Sections 5.3.1 and 5.3.2 for dataset descriptions and network training, respectively). Without the nonlinearity, these expressions are equivalent theoretically, as shown in (3.6) and (3.8). However, the placement of the nonlinearity σ can lead to differences in performance. While the performance of all the proposed convolution kernels are comparable overall, spectral

Table 3.3: Comparison of Different GSP Convolution Kernels

	Convolution Kernel	MUTAG	PROTEINS	IMDB-B	COLLAB
GCN	$\mathbf{A}x$	74.0 ± 7.3	72.8 ± 1.7	72.8 ± 2.3	79.5 ± 2.0
	$\mathbf{\Lambda}^* x$	77.1 ± 8.4	72.5 ± 1.8	71.2 ± 2.2	77.3 ± 1.8
	$\mathbf{\Lambda}\hat{x}$	$\mathbf{80.3 \pm 13.7}$	70.7 ± 1.7	70.6 ± 3.2	77.7 ± 0.8
	$\mathbf{M}\hat{x}$	71.3 ± 8.5	70.5 ± 6.3	63.0 ± 2.2	72.2 ± 1.3
	$GFT^{-1}\sigma(\mathbf{\Lambda}\hat{x})$	78.8 ± 8.4	69.9 ± 2.7	72.9 ± 2.7	76.1 ± 1.1
	$\sigma(GFT^{-1}\mathbf{\Lambda}\hat{x})$	79.3 ± 3.2	70.8 ± 1.7	69.9 ± 4.1	77.2 ± 1.0
TAGCN	$\mathrm{g}(\mathbf{A})x$	75.1 ± 8.2	72.4 ± 2.9	$\mathbf{73.3 \pm 5.3}$	$\mathbf{81.0 \pm 1.1}$
	$\mathrm{g}(\mathbf{\Lambda}^*)x$	78.7 ± 5.8	72.8 ± 2.6	72.1 ± 2.1	77.6 ± 1.1
	$\mathrm{g}(\mathbf{\Lambda})\hat{x}$	74.4 ± 7.9	$\mathbf{72.9 \pm 1.9}$	70.2 ± 2.3	77.7 ± 1.8
	$\mathrm{g}(\mathbf{M})\hat{x}$	73.4 ± 4.7	69.9 ± 3.2	72.5 ± 2.2	74.8 ± 1.0
	$GFT^{-1}\sigma(\mathrm{g}(\mathbf{\Lambda})\hat{x})$	76.1 ± 12.2	70.3 ± 2.2	72.2 ± 3.1	75.8 ± 1.3
	$\sigma(GFT^{-1}\mathrm{g}(\mathbf{\Lambda})\hat{x})$	78.2 ± 6.9	71.7 ± 2.5	70.3 ± 4.2	79.8 ± 0.1

domain approaches using $\hat{x}$ are better only for the MUTAG dataset when compared to simply using a polynomial of $\mathbf{A}$, $\mathrm{g}(\mathbf{A})$. This may be because the GFT data matrix, $\hat{x}$, and the spectral domain do not contain new information for classification.

3.3 Graph Pooling Layer

Pooling in some form is usually desirable in graph classification models for two reasons: dimensionality reduction and hierarchical learning. Graph pooling algorithms generally reduce the number of nodes and hence the number of learned parameters of the model. Some pooling algorithms also enforce hierarchical representation of the data, so the GNN can learn large-scale and global patterns in the data.

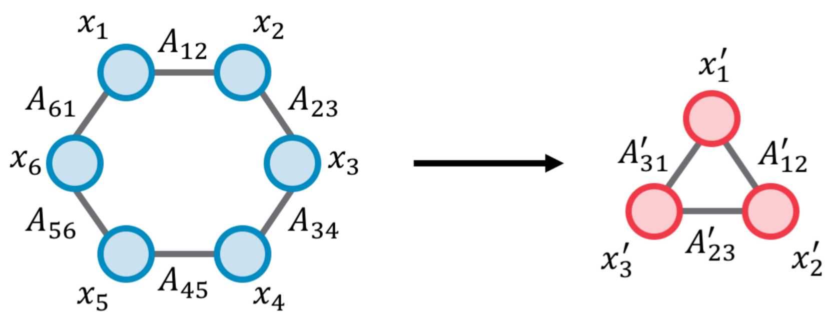


Figure 3.3: Graph pooling accepts a graph signal and produces a new, representative graph signal indexed by a smaller graph support. Image and caption taken from [2].

More formally, using the notation in Section 3.2, graph pooling yields signal $\mathbf{X}' \in$

$\mathbb{R}^{N' \times C}$ and adjacency matrix $\mathbf{A}' \in \mathbb{R}^{N' \times N'}$, with $N' \leq N$ (shown in Fig. 3.3). Pooling can be understood as a nonlinear downsampling operation. In deep learning, unlike as normally treated in DSP, recoverability is not a key concern for downsampling, and unlike graph convolution, which has a GSP definition, graph pooling has not been rigorously defined. In CNNs, max pooling is usually used, whereas in GNNs, there is no consensus how best to pool nodes in a graph. Recent methods of graph pooling include Sort Pooling (SortPool) [5], Differentiable Pooling (DiffPool) [3], Top-k Pool [6], and Self-Attention Graph Pooling (SagPool) [4]. See Fig. 3.4 for an overview. For a more in-depth discussion, see [51].

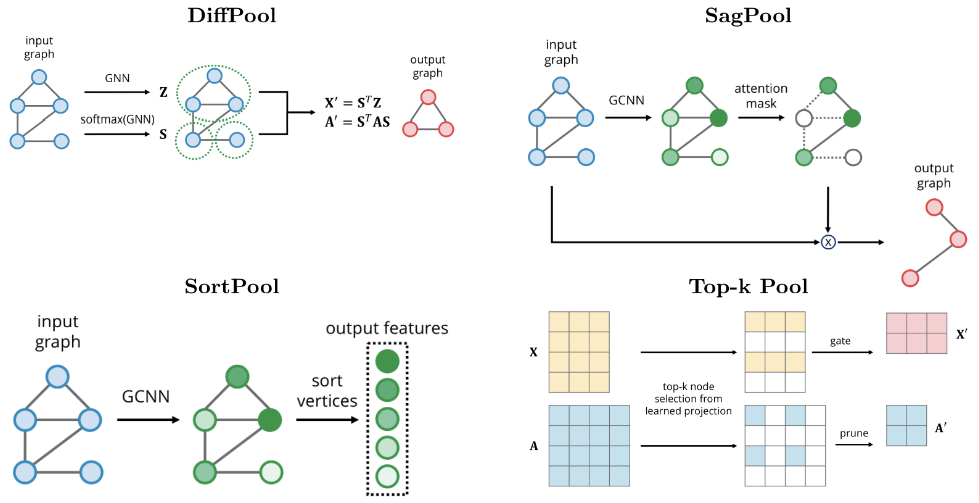


Figure 3.4: High-level illustrations of proposed graph pooling methods. DiffPool [3] uses a GNN model to obtain an assignment matrix for clustering the nodes. SagPool [4] uses a GCNN layer to calculate self-attention as mask for pooling the nodes. In SortPool [5], GNN layers are followed by a ranking of the nodes to select the top nodes. Top-k Pool [6] uses a projection vector to select the the top nodes, which form a new graph. Image and caption taken from [2].

3.4 Graph Aggregation Layer

In conventional CNNs, inputs are generally of the same size and have a fixed ordering. However, in graph classification problems, many graphs of various sizes are used with data defined on each graph in an arbitrary node permuted order (e.g., molecular networks). The resulting vectors that would serve as input to a fully connected layer vary in both size and labeling order, making direct comparison difficult.

The graph aggregation layer, also called the final pooling layer, solves this problem by collapsing the nodes into a fixed number of features, regardless of input size, for comparison. In general, this is done using a mean or sum operation over all nodes. However, several other statistics and approaches can be used. One such heuristic is the family of graph spectral distances, F_{GSD} [52], which uses just the adjacency matrix to capture global information about the graph structure by taking harmonic distances for all nodes.

Another potential approach is to use graph capsules. There are two variants of these: Graph Capsule Convolutional Neural Network (GCAPs-CNN) and Capsule Graph Neural Network (CapsGNN). GCAPs-CNN [53] uses a capsule vector that computes higher order statistics of each graph like 2nd and 3rd moments. CapsGNN [54] extracts node features first in the form of capsules, then graph features via an attention module by first concatenating all capsules across nodes.

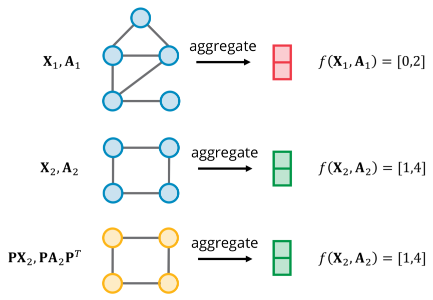

$\mathbf{X}_1, \mathbf{A}_1$
aggregate
$f(\mathbf{X}_1, \mathbf{A}_1) = [0,2]$
$\mathbf{X}_2, \mathbf{A}_2$
aggregate
$f(\mathbf{X}_2, \mathbf{A}_2) = [1,4]$
$\mathbf{PX}_2, \mathbf{PA}_2\mathbf{P}^T$
aggregate
$f(\mathbf{X}_2, \mathbf{A}_2) = [1,4]$

Figure 3.5: Diagram of global aggregation. Despite having different number of nodes (top and middle graphs), the aggregation produces vectors of the same size. Despite the nodes being permuted (middle and bottom graphs), the aggregation returns the same vector, $f(\mathbf{X}_2, \mathbf{A}_2) = [1, 4]$. Image and caption taken from [2].

Experiments Table 3.4 shows results of different aggregation methods including mean, variance, max, random selection of a node, F_{GSD} [52], GCAPS-CNN [53], and CapsGNN [54]. For all methods considered in the experiments, except F_{GSD}, graph signals are aggregated either using statistical quantities or attention mechanism. Results from F_{GSD} indicate that three (MUTAG, PROTEINS, IMDB-B) of the five datasets can be classified well just using the graph structure. In general, for GNNs, there is no single best way for aggregating across all the datasets, though certain aggregation methods may be complementary (e.g., combining mean and variance leads to an increase in accuracy). Sometimes, more complex techniques of aggregating may overfit the training data, e.g., combining mean and max reduces the test accuracy for the IMDB-B dataset. Techniques such as capsule based GNNs (e.g., GCAPS-CNN and CapsGNN) further improve performance for the MUTAG and PROTEINS datasets.

Table 3.4: Comparison of Aggregation Methods

	MUTAG	PROTEINS	IMDB-B	REDDIT-B	COLLAB
TAGCN (mean)	75.1 ± 8.2	72.4 ± 2.9	73.3 ± 5.3	91.6 ± 2.6	$\mathbf{81.0 \pm 1.1}$
TAGCN (var)	79.3 ± 4.2	73.5 ± 2.9	67.8 ± 2.3	$\mathbf{91.8 \pm 1.3}$	78.5 ± 0.9
TAGCN (max)	76.1 ± 5.5	73.0 ± 2.0	72.3 ± 2.7	90.3 ± 1.3	76.3 ± 2.0
TAGCN (random)	75.5 ± 1.1	67.1 ± 2.6	73.1 ± 2.8	85.8 ± 1.4	76.0 ± 1.7
TAGCN (mean+var)	76.1 ± 6.2	72.9 ± 2.4	74.0 ± 4.3	91.5 ± 1.7	79.5 ± 1.3
TAGCN (mean+max)	74.5 ± 8.3	74.6 ± 2.9	71.6 ± 2.7	90.6 ± 1.7	78.7 ± 2.6
F_{GSD} [52]	**92.1**	73.4	**73.6**	86.5	-
GCAPS-CNN [53]	-	$\mathbf{76.4 \pm 4.2}$	71.7 ± 3.4	87.6 ± 2.5	77.7 ± 2.5
CapsGNN [54]	86.7 ± 6.9	76.3 ± 3.6	73.1 ± 4.8	-	79.6 ± 0.9

3.5 Conclusion

To summarize, this chapter studies the GCNN architecture by considering its main blocks. We discuss two flavors of the graph convolutional layer: spectral and spatial (Section 3.2). Then we consider pooling (Section 3.3) and aggregation (Section 3.4) in detail, highlighting their design intuitions.

Chapter 4: Node Classification

One of the most common tasks in geometric deep learning is node classification where the algorithm infers the classes of unlabeled nodes in a graph by looking at the labeled nodes. Some examples include social networks and recommender systems [21]. One popular node classification benchmark uses citation networks, three of which are visualized in Fig. 4.1: CORA, CiteSeer, and PubMed [55]. In these graphs, each node represents a scientific publication, with bag-of-words feature vectors. An undirected edge is formed between two nodes if one cites the other. Node labels represent different publication categories. In CORA-ML, for example, the labels represent seven subfields of machine learning (e.g., computational biology and natural language processing). Geometric deep learning methods have achieved state-of-the-art over these and many other node classification problems [56, 21, 22].

In this chapter, we take a closer look as to why they perform well, by analyzing how data graph structure[1] affects the performance of these methods using real and synthetic datasets. For real datasets, we look at how simple metrics on real datasets can give clues to the performance of these methods and insights into selecting the appropriate architectures. The results motivate a more comprehensive investigation using synthetic datasets where we generate many graphs and analyze several model generation parameters/graph characteristics, e.g., edge creation probability of Erdős-Renỳi network model.

We organize the chapter as follows. We begin with the task definition for node classification (Section 4.1). Then we provide some background on the issues arising in node classification that we are addressing (Section 4.2) like over-smoothing (Section 4.2.4). Next, we break our analysis into two parts, using real datasets (Section 4.3), and synthetic datasets (Section 4.4). Finally, we summarize the results in the conclusions (Section 4.5).

[1]We refer to “data graph structure“ as “graph structure,“ and “data graph signal“ as “graph signal“ or “node feature.“

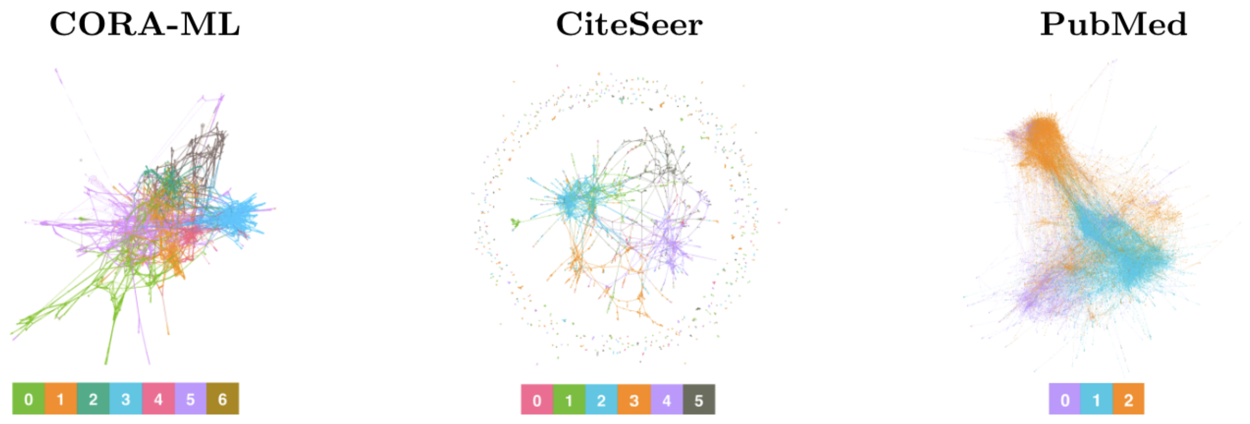


Figure 4.1: Visualization [7] of citation network datasets. Class labels are color-coded.

4.1 Task Definition

In node classification, we infer the classes of unlabeled nodes in a graph using the graph's labeled nodes. This is a semi-supervised task where generally only a small subset of nodes is labeled and used in training. It can be unsupervised if no labeled nodes are available for training. In this case, learning is achieved entirely through clustering or pooling nodes with similar features. The labels for the nodes can be categorical valued (binary or multiclass classification), or continuous valued (regression).

We now give a more formal definition following the notation defined in Section 2.1.1. Given a graph $\mathcal{G}$ and labels of a subset of the nodes $\mathcal{I} \subseteq [N]$, the task is to predict the labels of the remaining nodes $[N] \setminus \mathcal{I}$.

The graph $\mathcal{G}$ consists of a set of nodes V with $|V| = N$, signal $\mathbf{X} \in \mathbb{R}^{N \times C_0}$ on the nodes (where C_0 is the number of features on each node), a set of edges $\mathcal{E}$, and an adjacency matrix $\mathbf{A}$. $A_{ij} = 0$ unless there is an edge $e = (i, j)$ connecting node i and node j.

The goal is to learn a neural network function $f(\mathbf{X}, \mathbf{A})$ such that the prediction error between f and the true label $\mathbf{Y}$ is small. We use a non-negative real-valued loss function to model the prediction error:

$$L(f(\mathbf{X}, \mathbf{A}), \mathbf{Y}). \tag{4.1}$$

Now we can define the true risk associated with the function f by interpreting data as independent and identically distributed (i.i.d) random variables:

$$R_{\text{true}}(f) = \mathbb{E}_{\mathbf{X},\mathbf{A} \sim P}(L(f(\mathbf{X}, \mathbf{A}), \mathbf{Y})) = \int L(f(\mathbf{X}, \mathbf{A}), \mathbf{Y}) dP(\mathbf{X}, \mathbf{A}, \mathbf{Y}) \tag{4.2}$$

where P is the true distribution over the inputs. In practice, however, we do not have the true distribution P, so instead we use the empirical risk by averaging the loss function on the training set, i.e.,:

$$R_{\text{empirical}}(f) = \frac{1}{N_{\text{training}}} \sum_{i \in V_{\mathcal{I}}} L(f(\mathbf{X}_i, \mathbf{A}_i), \mathbf{Y}_i) \tag{4.3}$$

where $V_{\mathcal{I}}$ is the set of nodes available for training, and $N_{\text{training}} = |V_{\mathcal{I}}|$.

The principle of choosing the function f^* that minimizes this empirical risk is known as the empirical risk minimization:

$$f^* = \underset{f \in F}{\operatorname{argmin}} \, R_{\text{empirical}}(f) \tag{4.4}$$

where F is a fixed class of functions/neural networks under consideration. For GCNs and TAGCNs, this refers to the set of weights $\mathbf{W}^\star = (\mathbf{W^0}, \mathbf{W^1}, \ldots, \mathbf{W^L})$ and $\mathbf{W}^\star = (\mathbf{W^0_0}, \ldots \mathbf{W^0_K}, \mathbf{W^1_0}, \ldots, \mathbf{W^1_K}, \ldots \mathbf{W^L_K})$ respectively, as seen in (3.1) and (3.2), copied here. For GCN:

$$\mathbf{X}^{(\ell+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}^{(\ell)}\mathbf{W}^{(\ell)}\right) \tag{3.1}$$

For TAGCN:

$$\mathbf{X}^{(\ell+1)} = \sigma\left(\sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}^{(\ell)} \mathbf{W}_k^{(\ell)}\right). \tag{3.2}$$

We denote $\widehat{\mathbf{Y}} = f(\mathbf{X}, \mathbf{A})$ as the output of the neural network. For node classification, this can be a 2-layer TAGCN, and the training loss function is:

$$L(\widehat{\mathbf{Y}}_{\mathcal{I}}, \mathbf{Y}_{\mathcal{I}}) \tag{4.5}$$

where $\widehat{\mathbf{Y}}_{\mathcal{I}}$ corresponds to the logits predicted by the neural network during training (whose argmax corresponds to the predicted graph labels). For multi-class classification, we use the softmax cross-entropy loss:

$$L(\widehat{\mathbf{Y}}_{\mathcal{I}},, \mathbf{Y}_{\mathcal{I}}) = -\frac{1}{N_{\text{training}}} \sum_{v \in V_{\mathcal{I}}} \sum_{i \in N_c} \log(S((\widehat{\mathbf{Y}}_v)_i))(\mathbf{Y}_v)_i \tag{4.6}$$

where N_C is the number of classes and $(\mathbf{Y}_v)_i$ is the one-hot encoding of the class of the true label (all 0 except 1 for each label). The softmax function $S(\cdot)$ used to convert the logits to probabilities is:

$$S((\widehat{\mathbf{Y}}_v)_i) = \frac{\exp((\widehat{\mathbf{Y}}_v)_i)}{\sum_{j=1}^{N_c} \exp((\widehat{\mathbf{Y}}_v)_j)}. \tag{4.7}$$

We use the above metrics during training and validation. For testing, we report the performance based on the test accuracy (using argmax to select the predicted class $\widehat{\mathbf{Y}}_{\text{testing}}$).

4.2 Issues Considered

In this section, we present the main issues that we address using real and synthetic datasets.

4.2.1 Impact of the Graph Structure

The primary issue that we are addressing is how the graph structure affects the performance of graph neural networks. We study how the performance changes if different graph structures are used. Our experiments show that, in general, incorporating the relevant graph structure over a random/identity graph structure into training improves the performance. In Fig. 1.2 in the introduction, we show that using the neural network model using graph structure significantly outperforms the same model when the graph structure is ignored, i.e., trained with the identity matrix.

We also study how node degrees affect the performance of GNNs. In general, nodes with large degrees have significant impact (since they connect with more nodes):

$$\text{Impact}_{V_i} = \frac{\deg(V_i)}{\sum_{v \in V} \deg(v_i)} \tag{4.8}$$

In our studies, when using real datasets, we examine the relationship between test accuracy and network statistics and metrics. When using synthetic datasets, we investigate connections between the test accuracy and graph generation parameters, e.g., interclass and intraclass edge creation probability for Erdős-Rènyi graphs.

4.2.2 Impact of Graph Signal

Although generally not the case, there can be times when the graph signal/feature is missing, uninformative, or adversarial. An adversarial case is when the features are the same for some classes, and different for others (so only helpful for some nodes). In cases where the graph signal or features are missing, PyTorch Geometric [57] uses the degree of the nodes as features.

We also investigate this using synthetic data by generating various graph signals and consider them in combination with different graph structures.

4.2.3 Impact of the Graph Neural Network Architecture

Alongside our above objective, we also study how the architecture of the graph neural network affects the test performance. We consider two architectures: graph convolution networks (GCNs) [17] and topology adaptive graph neural networks (TAGCNs) [48]. We look into hyperparameters like the number of layers for GCN and TAGCN (defined in 3.2). For the latter, we also consider the degree of the polynomial filters.

For both real and synthetic datasets, we study the architecture in conjunction with the graph structure. We also consider simple variants of these methods that can better capture some of the real graph characteristics and synthetic graph generation parameters, hence improving performance.

4.2.4 Over-smoothing

One important issue when applying graph neural networks, especially in node classification, is over-smoothing. It arises as a result of trying to increase the expressiveness of the graph structure by increasing GNN depths. While adding more layers leads to broader receptive fields of nodes, it may also lead to a model that treats the nodes equally. In other words, there is a trade-off in expressiveness between having far away nodes that are able to communicate and having more distinguishable representations of nodes in different classes. In general, we want smoothing of nodes of the same class but not of the different classes.

More formally, [58] showed that graph convolution is a form of Laplacian smoothing. The standard form is:

$$(\mathbf{I} - \gamma \widetilde{\mathbf{D}}^{-1} \widetilde{\mathbf{L}})\mathbf{X} \tag{4.9}$$

where $\widetilde{\mathbf{D}} = \sum_j \widetilde{\mathbf{A}}_{ij}$, $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, and $\widetilde{\mathbf{L}}$ is the Laplacian $\widetilde{\mathbf{D}} - \widetilde{\mathbf{A}}$.

With GCN [17], if we let $\gamma = 1$ and replace $\widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{L}}$ with $\widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{L}}\widetilde{\mathbf{D}}^{-1/2}$, we get:

$$(\mathbf{I} - \gamma \widetilde{\mathbf{D}}^{-1/2} \widetilde{\mathbf{L}} \widetilde{\mathbf{D}}^{-1/2})\mathbf{X} \tag{4.10}$$

$$= (\widetilde{\mathbf{D}}^{-1/2} \widetilde{\mathbf{A}} \widetilde{\mathbf{D}}^{-1/2})\mathbf{X} \tag{4.11}$$

If we iteratively apply this convolution and if the graph is connected, the node features converge to a steady state.

Theorem 1. *If a graph is connected, then for any $x \in \mathbb{R}^n$:*

$$\lim_{m \to \infty} (\widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2})^m \mathbf{x} = \mathbf{D}^{-1/2}\theta$$

where $\theta \in \mathbb{R}^{N \times 1}$ is independent of x.

Proof. (Sketch) The eigenvalues of the normalized Laplacian $\widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{L}}\widetilde{\mathbf{D}}^{-1/2}$ fall between 0 and 2 [59]. The eigenspaces corresponding to eigenvalue of 0 are spanned by $\mathbf{D}^{-1/2}$ [60]. Hence, the eigenvalues of $\mathbf{I} - \widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{L}}\widetilde{\mathbf{D}}^{-1/2}$ fall into $(-1, 1]$ and the eigenspace of eigenvalue 1 is spanned by $\mathbf{D}^{-1/2}$. Since the absolute value of all eigenvalues is less than or equal to 1, repeated multiplication converges to a linear combination of eigenvectors of 1, which is $\mathbf{D}^{-1/2}$. See [58] for the original proof. □

So the result is only a linear combination of the graph structure, resulting in the loss of discriminative power. Similar results apply to TAGCN, but the effect is not as severe due to residual connections.

There are many ways to quantify and reduce over-smoothing [61, 62, 63]. The focus here, instead, is to analyze it in more detail experimentally with respect to not just the number of layers, but also to other architectural design choices (like order of polynomial filters) as well as the graph structure. We present results in relation to these elements on real and synthetic datasets.

4.2.5 Rate of Training Convergence

Another potential issue is the rate of convergence in training. For CORA, a graph with 2485 nodes, the training time per cross-validated fold for a 2-layer GCN and 2-layer 3-Degree TAGCN on a NVIDIA GeForce RTX 2080 Ti using Deep Graph Library is on average 0.52 seconds and 1.03 seconds, respectively. However, for much larger graphs, this can be a potential issue in terms of time and computational resources. For example, if the graphs are dense/complete, the default sparse message passing implementation for matrix multiplication in most frameworks is not efficient and use more GPU/CPU memory than dense matrix multiplication (which is in general not efficient for sparse graphs). Coauthor Physics dataset is too large to be loaded using minimum TAGCN architecture. On the architecture side, larger neural networks (greater number of layers, higher degree of the polynomial filter) leads to more training time per epoch (due to having more weights to train). However, they can accelerate the training by reducing the convergence rate.

Theoretical and experimental analysis on how the architecture and the data graph can affect the convergence rate is discussed in chapter 6. In Chapter 6, Theorems 2, 3, 4 describe the relation between the convergence to global minimum for 1-Layer + 1-ordered TAGCN, 1-Layer + K-ordered TAGCN, and L-Layers + K-ordered TAGCN (ordered referring to the degree of the polynomial filter).

4.2.6 Evaluation Setup

Another major issue is the evaluation setup. According to [64], using the same train/validation/test splits and different training procedure (e.g., early stopping criteria) underestimates the generalization errors and precludes a fair comparison of different architectural design choices. Reference [64] found that GCN [17], the simplest model, actually performs the best across many node classification datasets when using random splits and systematic training procedure. For both real and synthetic datasets, we use random splits and the same training procedure for all GNN architectures and hyperparameters. For synthetic datasets, we generate 50 random graphs for each set of network model parameters and average the results.

4.3 Real Data

In this section, we analyze the impact of graph structure on the performance of GNNs for 7 real node classification datasets. We divide the section into 3 parts: datasets description and their statistics, experimental setup, and results.

4.3.1 Data Description and Statistics

Table 4.1 lists the common node classification benchmarks that we are using and their data statistics. Table 4.2 shows the network metrics where % of intraclass and % interclass edges are the number of intraclass and interclass edges divided by the total number of edges.

CORA [65], CiteSeer [66], and PubMed [67] are citation network datasets. Nodes are scientific papers and edges represent citations between papers. Node feature is a bag-of-words vector. Labels indicate the field of study.

Amazon Computers and Amazon Photos are segments of the Amazon co-purchase graph [68]. Nodes represent items, and edges represent two items commonly purchased

Table 4.1: Node Classification Data Statistics [1]

Dataset	Nodes	Edges	Features	Classes	Label Rate	Edge Density
CORA	2485	5069	1433	7	0.0563	0.0016
CiteSeer	2120	3705	3703	6	0.0566	0.0016
PubMed	19717	44324	500	3	0.0030	0.00023
Coauthor CS	18333	163788	6805	15	0.0164	0.00097
Coauthor Physics	34493	247962	8415	5	0.0029	0.00042
Amazon Computer	13381	245778	767	10	0.0149	0.0027
Amazon Photo	7650	288163	745	8	0.0209	0.0098

at the same time. Node feature is bag-of-words vector taken from product reviews. Label is the category of the item.

Coauthor CS and Coauthor Physics are co-authorship graphs from KDD Cup 2016 challenge[2]. Nodes are authors and edges represent coauthorship. The feature is the paper keywords for each author's papers. Labels represent each author's most active fields of study.

Table 4.2: Node Classification Data Metrics [1]

Dataset	Average Degree	% of Intraclass Edges	% of Interclass Edges	Difference in %
CORA	2.04	80.41	19.59	60.82
CiteSeer	1.75	73.66	26.34	47.31
PubMed	2.25	80.24	19.76	60.48
Coauthor CS	8.93	80.81	19.19	61.61
Coauthor Physics	7.19	93.14	6.86	86.28
Amazon Computers	18.37	77.72	22.28	55.44
Amazon Photo	37.67	82.72	17.28	65.44

4.3.2 Experimental Setup

Following the standardized experimental setup discussed in Section 4.2.6, we use the same procedure for all models and data. We use Adam optimizer with learning of 0.001, and default initialization using (weights initialized using Glorot and biases initialized with zeros), 10^{-3} L_2 regularization/learning decay, up to 100k epochs with early stopping condition of total validation loss not improving in 50 epochs (so actual training time is much shorter). We split the data with 100 random train/validation/test splits and 20 random initializations. We use full-batch training. For the GNNs, we perform

[2]https://kddcup2016.azurewebsites.net

extensive grid search for hidden channels (8, 16, 32, 64), number of layers (1-10), and for TAGCN, degree of the polynomial filters (1-5) unless otherwise specified. Except for the last layer, we apply a dropout rate of 0.5 between each GNN layer. We use softmax cross-entropy loss. For each model, we report the test accuracy picked using the lowest validation loss across all hyperparameters. We use 20 and 30 labeled nodes per class as the training and validation set respectively with the rest as the test set.

4.3.3 Results

We report the results of several elements mentioned in Section 4.2 on model performance for 7 real node classification datasets, quantified by test accuracy.

4.3.3.1 Impact of Graph Structure

Since the graph structure is fixed on each dataset, we compare the results to using no graph structure. The identity matrix case shown in Fig. 1.2 is analogous to ignoring the graph structure and applying a multilayer perceptron (MLP) to the signals. Table 4.3 shows the test accuracy for GNNs on graphs vs MLPs on graph signals only. In general, more intraclass edges implies more meaningful graph structure, while more interclass edges implies less meaningful graph structures. For graphs with high average degree and high difference in percentage between intraclass and interclass edge, we expect not just higher accuracy vs using no graph structure, but also more resistance to over-smoothing (discussed in Section 4.2.4).

As we have a very small sample size (7 samples), we are not able to make any strong conclusion. However, we have some observations that support our theory. Amazon Computers and Amazon Photos have a large difference in perctange of intraclass - interclass of edges along with very high node degrees. They also have large difference in test accuracy between using and not using the underlying graph structure.

Simply having a large number of intraclass edges and high average degree does not necessarily lead to improvement in performance. This may be the result of the graph signals. The most noticeable example is the Coauthor Physics dataset, which has a very high difference in percentage of intraclass to interclass, but only minimal improvement with GNNs over MLPs. We hypothesize that this is because it has much better features (good performance using just MLPs), so the performance improvement with graph structure is minimal.

Although not illustrated here, it may be possible to have graph structures that lead to worse performance than just the identity. For example, when having a much higher percentage of interclass edges than intraclass edges. In this case, GNNs learn mostly from the graph signal and disregard the graph structure by assigning weights for the non-zero polynomial filters to 0.

Table 4.3: Node Classification Results of Graph (GNNs) vs Without Graph (MLPs)

	GCNs	TAGCNs	MLPs	Difference of GCNs and MLPs
Cora	77.4 +/- 2.4	79.7 +/- 1.5	57.9 +/- 3.5	19.5
Citeseer	69.0 +/- 1.1	67.7 +/- 0.8	59.1 +/- 1.3	9.9
PubMed	76.1 +/- 2.3	77.8 +/- 1.6	71.9+/- 2.2	4.2
Coauthor CS	90 +/- 0.5	91.1 +/- 1.5[3],	87.0 +/- 1.9	3.0
Coauthor Physics	91.9 +/- 1.0	N/A [3]	87.8 +/- 1.5	4.1
Amazon Computers	80.3 +/- 1.1	77.7 +/- 2.7	66.5 +/- 2.8	13.8
Amazon Photo	90.1 +/- 0.8	85.6 +/- 2.4	77.4 +/- 2.2	12.7

4.3.3.2 Impact of Graph Neural Network Architecture

In this section, we analyze two aspects of graph neural network architectures: number of layers and degree of the polynomial filters.

GCN: Number of Layers Fig. 4.2 shows the results of number of layers vs test accuracy using GCN for real datasets. The optimal accuracy for all the datasets occurs in the 1st or 2nd layers. In general, accuracy tends to drop as the number of layers increases due to over-smoothing (see Section 4.2.4). The accuracy is random when it over-smoothes, i.e., $\frac{1}{\text{number of classes}}$, e.g., 33% for PubMed, 14% for Cora, based on the number of classes (see Table 4.1). The largest drop in accuracy is generally accompanied by large variance, indicating initializations and data splits have an effect on over-smoothing. For datasets like PubMed, under some initial conditions, GCN over-smooths, for some others, it does not. In general, it is more likely to over-smooth if the interclass features of the training set are similar.

For datasets with high difference in % of intraclass and interclass edges like Coauthor Physics, the effect of over-smoothing is less. As shown in Fig. 4.2, Amazon Computers is

[3]Data (graph structure and signal) is too large to be loaded onto the GPU using TAGCN architecture, and requires neighborhood sampling, which would make comparison unfair. For Coauthor CS dataset, we only consider polynomial filters up to degree 3. For Coauthor Physics, it requires more than 10 GB even for a single layer.

hovering at around 55% accuracy even at 9 and 10 layers (with high variance), whereas Coauthor Physics has not dropped even at 10 layers. They do eventually over-smooth at much higher number of layers.

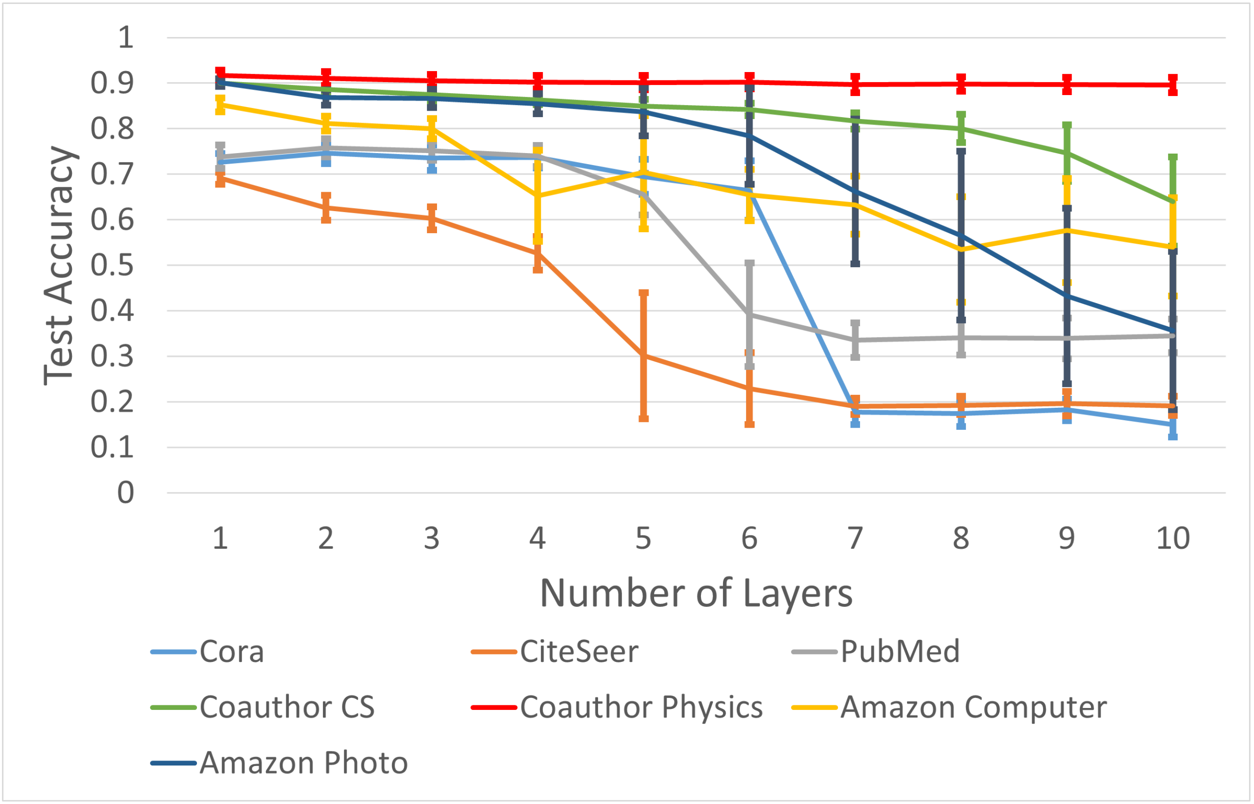


Figure 4.2: GCN: Test accuracy vs number of layers for 7 real datasets. Colored horizontal and vertical lines represent average test accuracy and standard deviation of each dataset, respectively.

TAGCN: Degree of the Polynomial Filters Fig. 4.3 shows the impact of the degree of the polynomial filters on these datasets for 1-layer and 2-layers TAGCN[4]. For 1-layer, the degree of the polynomial filters does not have a significant effect on citation networks (Cora, CiteSeer, PubMed) and Coauthor CS dataset. Higher degree has a significant effect on Amazon Computers and a slight effect on Amazon Photo dataset. Based on Table 4.2, this might be due to their high average degree. There is significant overfitting for all the datasets (more than GCN), with training accuracy approaching 100% due to model complexity despite regularization with higher dropout rates and weight decay

[4]Coauthor CS's graph is too large and uses too much memory for 2-layer TAGCN with degrees of the polynomial filters greater than 3, and cannot be completely loaded onto the GPU. A stochastic TAGCN that samples nodes for convolution is possible, but the comparison would not be fair.

due. For 2-layers, degree of the polynomial filters has more impact on the Amazon Computers and Amazon Photos dataset, with a sudden drop in performance after the 1st degree polynomial filter. Higher degree of the polynomial filters leads to significant deterioration in performance (close to random after polynomial filters of degree 5), indicating significant over-smoothing.

TAGCN: Number of Layers and Degree of the Polynomial Filters Fig 4.4 shows the results of TAGCN where we observe both the number of layers and the degree of the polynomial filters for 6 of the datasets[5]. In general, having small number of layers regardless of the degree of the polynomial filters results in the highest accuracy for all datasets. There is no clear result as to the degree of the polynomial filters for different datasets and number of layers. The variations seem to be small relative to number of layers. Using degrees of the polynomial filters up to 5 also does not eliminate the effects of over-smoothing, as accuracy drops significantly to random accuracy at high number of layers. For Amazon Photos dataset, 5 seems to be the optimal degree of the polynomial filters for most layers, whereas for Amazon Computers dataset, it seems to be 1.

4.3.3.3 GNN Variants

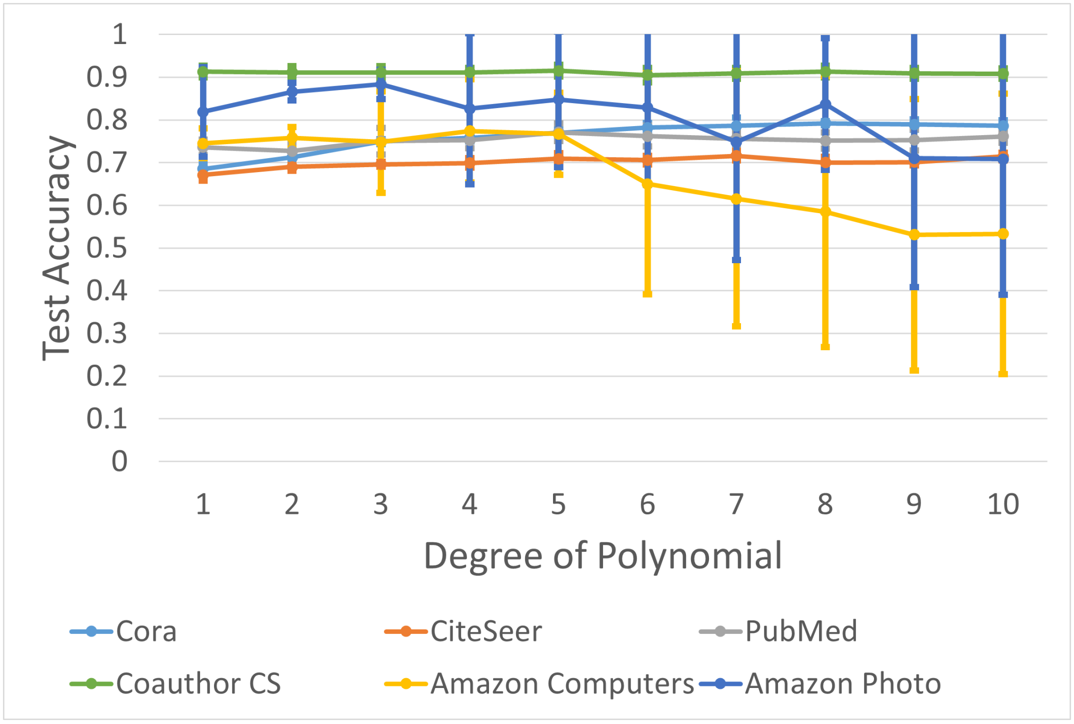
We consider three GNN variants: degree-aware GCNs (DA-GCNs), degree-aware TAGCNs (DA-TAGCNs), and graph attention networks (GATs) [50]. DA-GCNs and DA-TAGCNs, defined formally in Section 5.3.3.3, use a trainable weight for each degree of the node (for TAGCN, each degree of the polynomial filters). This is effective if the node degrees of different classes are distinct.

GATs employ self-attention over the node features and define an edge weight based on its self-attention score (normalized over all its neighbors). This is effective if the weights of the adjacency matrix are uninformative while the graph signal is informative. For example, when the weights are randomly generated or all 1's and when the signals are distinct for different classes. However, if the signals are not distinct for different classes, GATs can produce a less effective graph structure (and hence worse results).

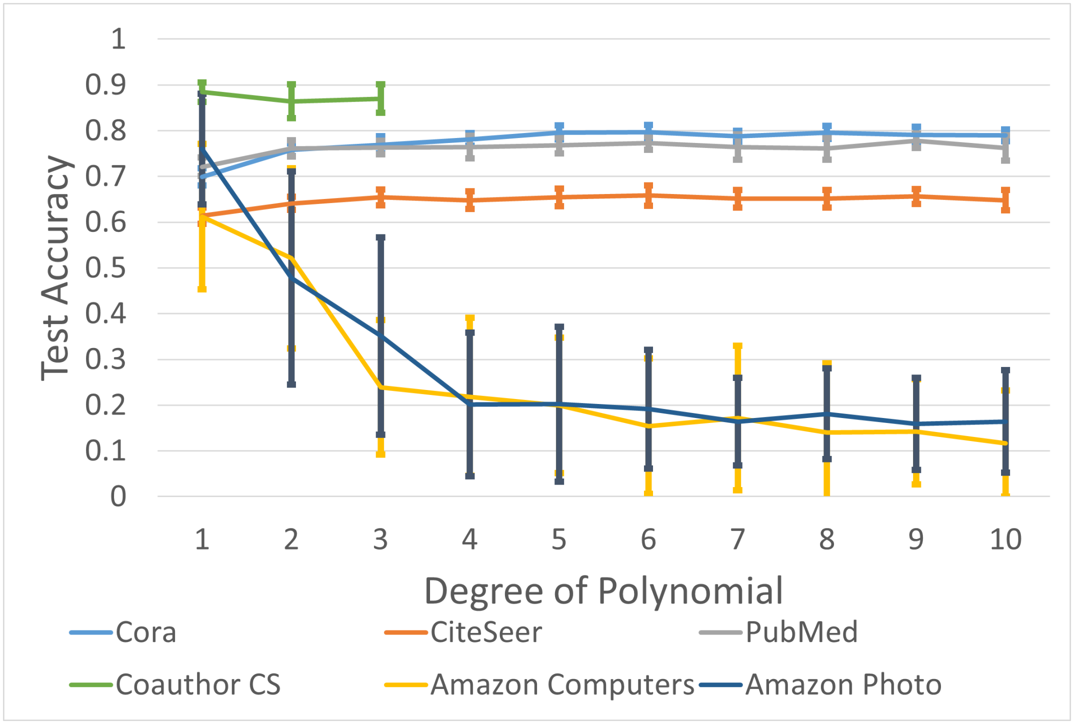
Fig. 4.5 shows the results of GCN, DA-GCN, TAGCN, DA-TAGCN, and GAT[6]. In general, GCN and TAGCN perform better than DA-GCN and DA-TAGCN, implying that the degrees of the nodes are not very distinct for different classes. The only notable

[5]Coauthor Physics's graph is too large and uses too much memory for even 1-layer TAGCN, and cannot be completely loaded onto the GPU we have access to.

[6]Coauthor Physics' graph is too large and uses too much memory for a fair comparison.

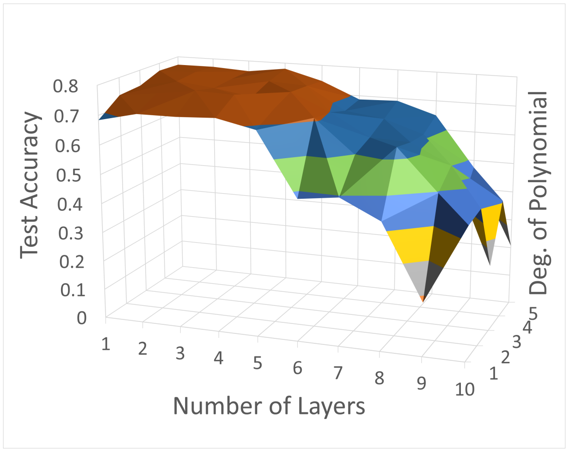


(a) 1-layer TAGCN

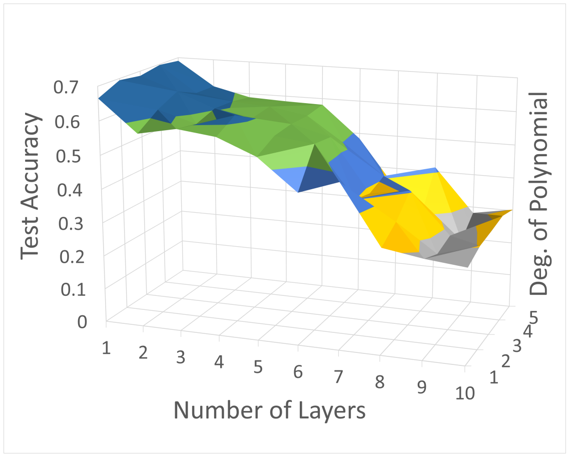


(b) 2-layer TAGCN

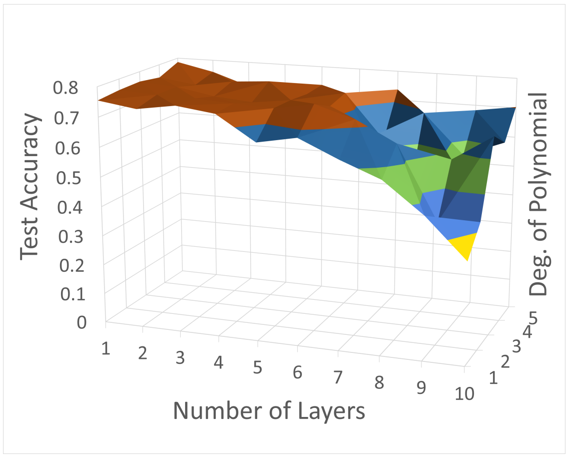
Figure 4.3: TAGCN: Test accuracy vs degree of the polynomial filters with 1-2 convolutional layers for 6 real datasets. Colored horizontal and vertical lines represent average test accuracy and standard deviation of each dataset, respectively.



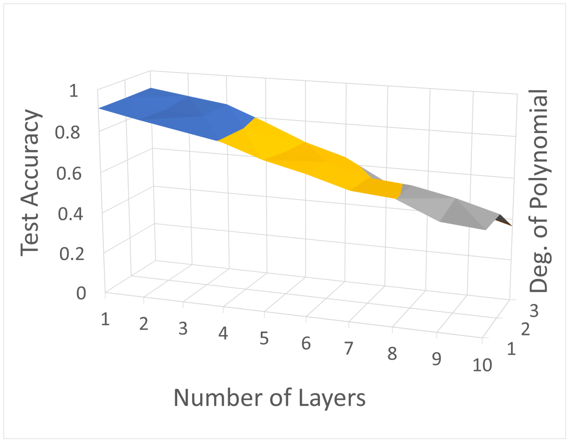
(a) Cora



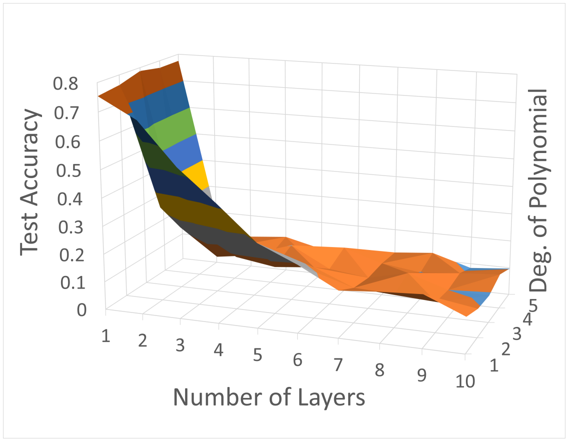
(b) Citeseer



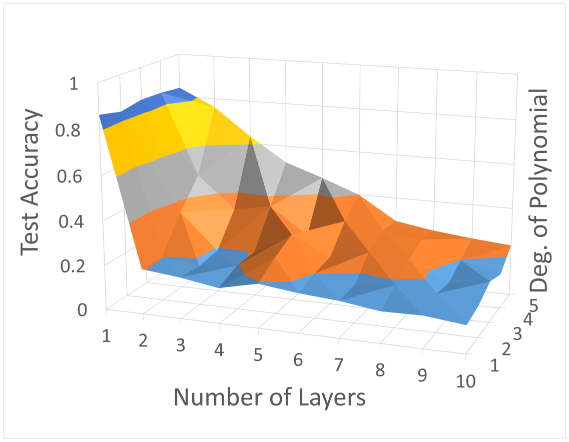
(c) PubMed



(d) Coauthor CS



(e) Amazon Computers



(f) Amazon Photos

Figure 4.4: TAGCN: Test accuracy vs number of layers and degree of the polynomial filters for 6 real datasets.

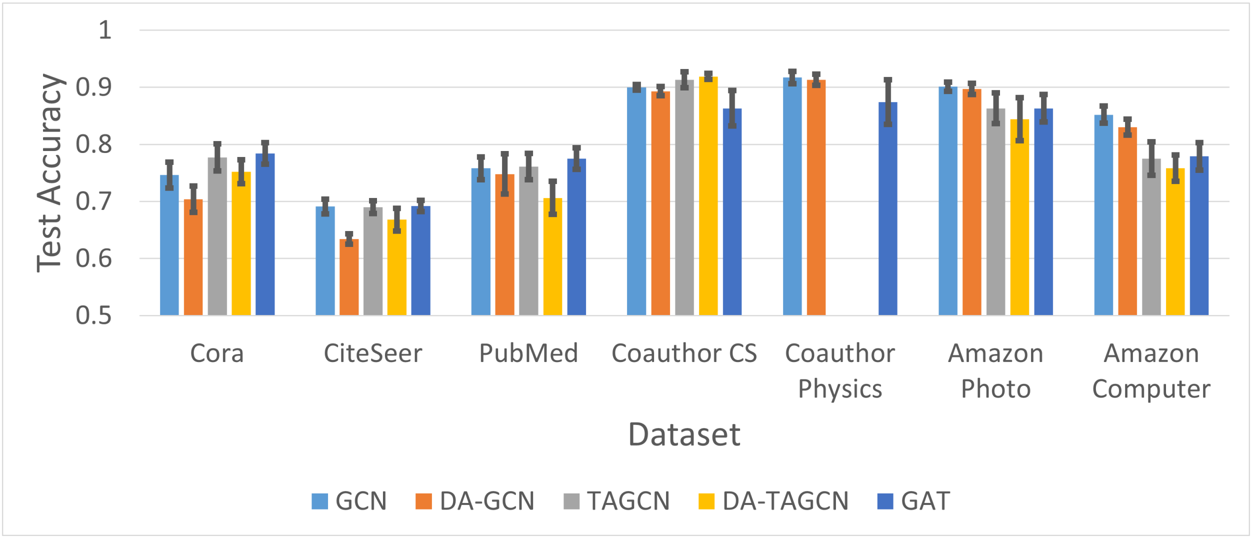


Figure 4.5: GCN and TAGCN vs GNN Variants (DA-GCN, DA-TAGCN, GAT) for 7 real datasets. Colored bars represent average test accuracy of each architecture. Vertical black lines on the bars represent standard deviation.

difference is the Coauthor CS dataset, which shows a slightly higher test accuracy with DA-TAGCN than with TAGCN. It is also one of the few datasets that have high test accuracy when using only the degree of the node for classification, relative to the number of label categories (the other is Amazon Computer).

Another surprising finding is that GAT performs worse on all datasets except CORA and PubMed. This suggests that only CORA and PubMed have graph signals useful for learning edge weights. Another potential reason is the evaluation setup discussed in Section 4.2.6 (since [50] and others report higher accuracy using GAT). A closer look at the results indicate that GAT scores very low for some weight initializations and splits (around $\frac{1}{20}$ of the times).

4.4 Synthetic Data

Compared to real datasets, synthetic datasets allow for a more comprehensive analysis of the impact of graph structure on the performance of the GNN architecture topology. This is because 1) with synthetic datasets, we can systematize the graphs and study the impact of each characteristic of the graph at a time, and 2) we can generate many dataset samples to draw more reliable conclusions.

4.4.1 Data Description

We focus on 3 models of graphs that are commonly studied: Erdős-Rényi, smallworld, and preferential attachment.

4.4.1.1 Erdős-Rényi

$\mathcal{G}(n, p)$: n nodes with edge creation probability p. We define an additional parameter r with $p = \frac{r \ln n}{n}$. We set $r > 1$ so $\mathcal{G}(n, p)$ is almost surely connected. Erdős Rényi networks have low clustering. They are a popular graph model for generating random graphs. They have a binomial degree distribution [69]:

$$P(\deg(v) = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}. \tag{4.12}$$

4.4.1.2 Smallworld

Watts–Strogatz model[70]: $\mathcal{G}(n, \kappa, p)$: n nodes with mean degree κ and rewiring probability p. Each node is connected to its $\frac{\kappa}{2}$ nearest neighbors on each side in a undirected ring topology. For every node, consider each of the edges to its $\frac{\kappa}{2}$ rightmost neighbors, rewire it to any node with probability p, while avoiding self-loops. Watts–Strogatz graphs generally have high clusterisation and short path length. These small-world properties are common in nature (e.g., seismic activity, brain connectivity). The degree of the network has a Dirac delta distribution [71]:

$$P(\deg(v) = j) = \sum_{i=0}^{f(j,\kappa)} \binom{\kappa/2}{i} (1-p)^i p^{\kappa/2-i} \frac{(p\kappa/2)^{j-\kappa/2-i}}{(j-\kappa/2-i)!} e^{-p\kappa/2}. \tag{4.13}$$

for large n and $f(j, \kappa) = \min(j - \kappa, \kappa/2)$.

4.4.1.3 Preferential Attachment

Barabási–Albert model [72]: $\mathcal{G}(n, m)$: n nodes are grown by attaching new nodes with m (edges-to-attach) edges with a preference to nodes with high degree, i.e., the probability of creating an edge to node i is: $p_i = \frac{k_i}{\sum_j k_j}$ where k_i is the degree of node i. Barabási–Albert graph models are motivated by networks that are approximately scale-free and contain few nodes with high degree (e.g., social networks, citation networks). The network has a power law degree distribution, for example [73]:

$$P(\deg(v) = k) = \frac{0.82}{k^3}. \tag{4.14}$$

4.4.2 Experimental Setup

We follow similar experimental setup as with the real datasets (See section 4.3.2). For default graph generation setup, we find parameters that lead to distinguishable results. Unless mentioned otherwise, we use:

- 2 classes, 0 and 1
- 1000 Nodes, 500 for each class
- 50 random graphs from each set of network model parameters
- Graph signal x is a 1-dimensional feature vector with $N(\mu_{Y_i}, 1)$ where Y_i is the class of node i. The class means μ_0 and μ_1 are taken to be $\mu_{\text{signal}} = \pm 0.1$.
- For Erdős-Rényi, we generate single graph with the following intraclass and interclass edge creation probability: $p_{\text{intraclass}} = 0.07$ and $p_{\text{interclass}} = 0.03$.
- For Watts–Strogatz, we generate 2 separate Watts–Strogatz graphs with the same mean degree $\kappa_1 = \kappa_2 = 50$ and assign the nodes of each graph to different classes. For all edges, we apply an edge rewiring probability with $p_{\text{intraclass}} = 0.1$ to edge of the same class, and $p_{\text{interclass}} = 0.1$ to edge of different class.
- For Barabási–Albert, we generate 2 separate Barabási–Albert graphs with the same edges-to-attach $m = 30$ (each new node forms m links to existing nodes). Then we apply an edge rewiring probability with $p_{\text{interclass}} = 0.3$ to all edges. For each rewiring, we remove the existing edge, and connect the source node to a node of the opposite class, prioritizing nodes of higher degree (so keeping the preferential attachment). The priority probability is the same as Barabási–Albert edge connection probability, but just for a different class, i.e., the probability that the rewired node is connected is connected to node i of the opposite class is $p_i = \frac{\deg(i)}{\sum_{j \in \text{nodes in opposite class}} \deg(j)}$.

4.4.3 Results

Here, we report the results of several characteristics on model performance for 3 models of synthetic graph classification networks, quantified by test accuracy. For the architecture, we focus on the number of layers and the degree of the polynomial filters.

4.4.3.1 General Results

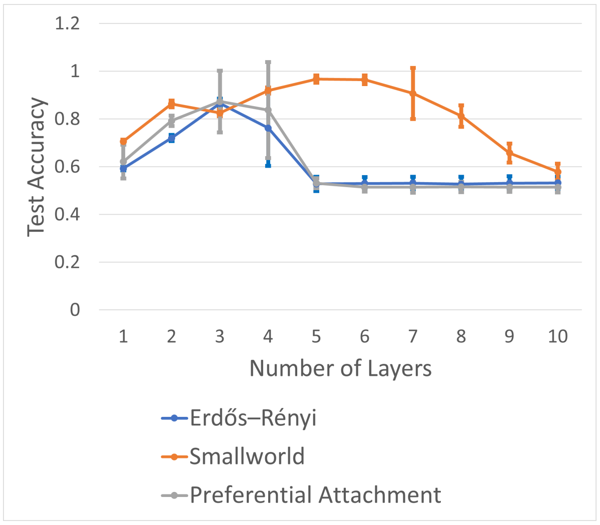
We first discuss general results comparing the 3 network models. Fig 4.6a shows the results of GCN for different number of layers. Although all networks succumb to over-smoothing, smallworld seems to be the most resistant, with plateau from layers 4-7, before a gradual descent to 50% accuracy at layer 10. For all networks, there is high variance in performance when over-smoothing starts to occur. Accuracies start to drop due to over-smoothing at layers 4, 7, and 4 for Erdős-Rényi, smallworld, and preferential attachment, respectively.

Fig. 4.6b, 4.6c, and 4.6d show the results of TAGCN for different number of layers and degree of the polynomial filters on the 3 models of synthetic graphs. The degree of the polynomial filters seems to have different effect for different network. For Erdős-Rényi (Fig. 4.6b), 3 seems to be the optimal degree of polynomial filters (peak occurs at layer 4, order 3). For smallworld (Fig. 4.6c), the filters play major role in optimizing the performance and accelerating the effect of over-smoothing for the smallworld dataset, with significant drop in accuracy only when the sum of the number of layers and the degree of the polynomial filters is greater than 8 for the 1st 3 layers. For preferential attachment (Fig. 4.6d), it seems to have little effect vs number of layers.

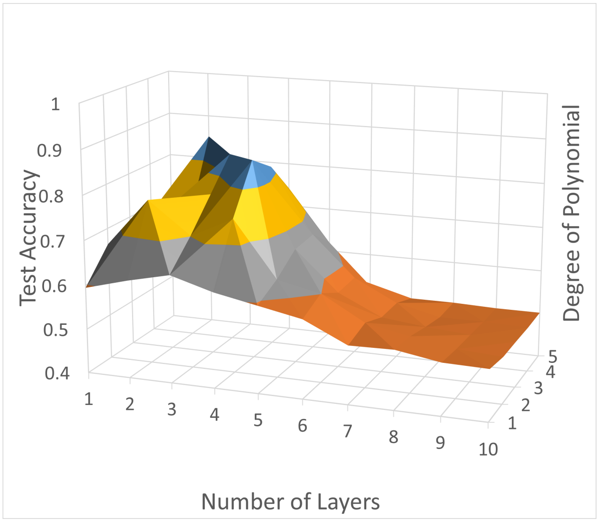
4.4.3.2 Erdős-Rényi Network Results

Fig. 4.7 and 4.8 show the results of GCN and TAGCN vs number of layers, number of layers + degree of the polynomial filters, respectively, for different Erdős-Rényi model generation parameters, which are intraclass edge creation probability $p_{\text{intraclass}}$, interclass edge creation probability $p_{\text{interclass}}$, interclass difference in means of graph signals, and number of nodes.

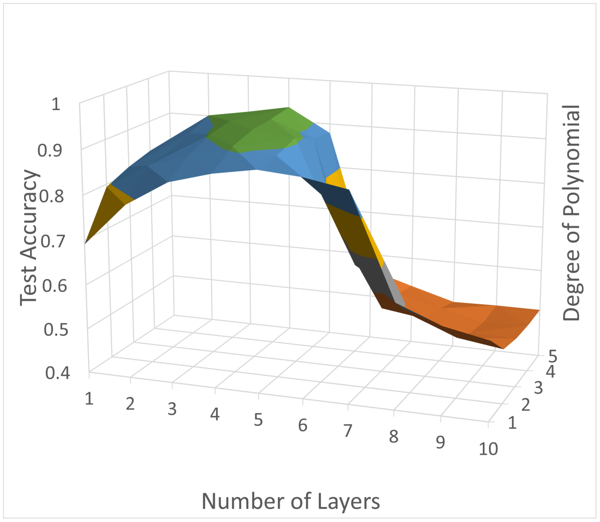
Impact of Intraclass Edge Creation Probability $p_{\text{intraclass}}$ Fig. 4.7a shows the results of GCN vs number of layers for different intraclass edge creation probability. In general, as the intraclass edge creation probability increases, the test accuracy improves. When the



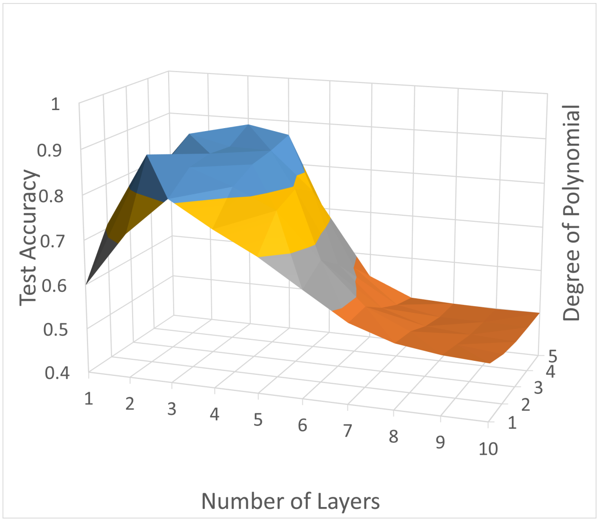
(a) GCN: All 3 models of synthetic graphs. Colored horizontal and vertical lines represent average test accuracy and standard deviation of each dataset, respectively.



(b) TAGCN: Erdős-Rényi

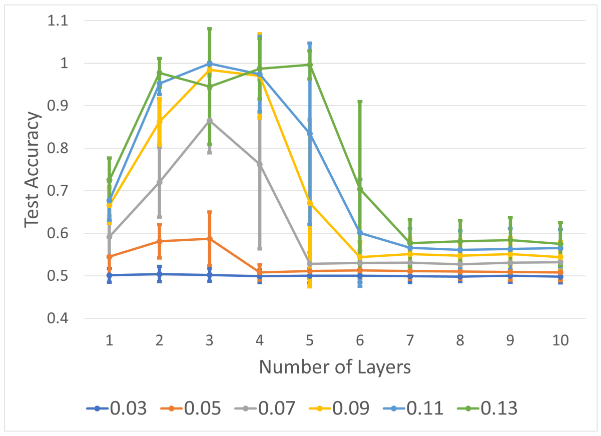


(c) TAGCN: Smallworld

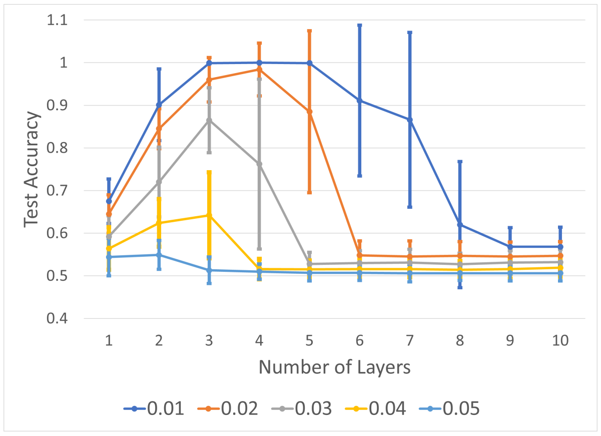


(d) TAGCN: Preferential Attachment

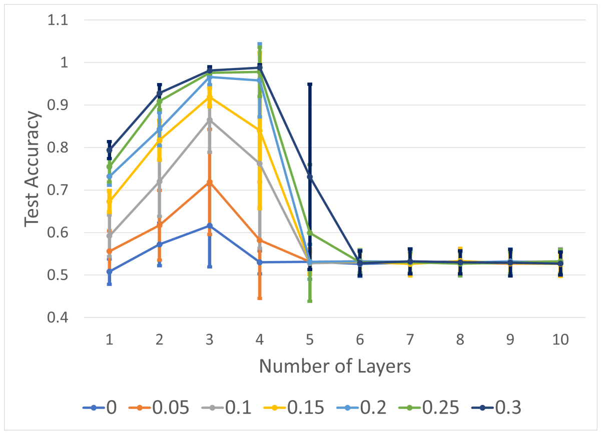
Figure 4.6: General GCN and TAGCN: Test accuracy vs number of layers and/or degree of the polynomial filters for 3 models of synthetic graphs.



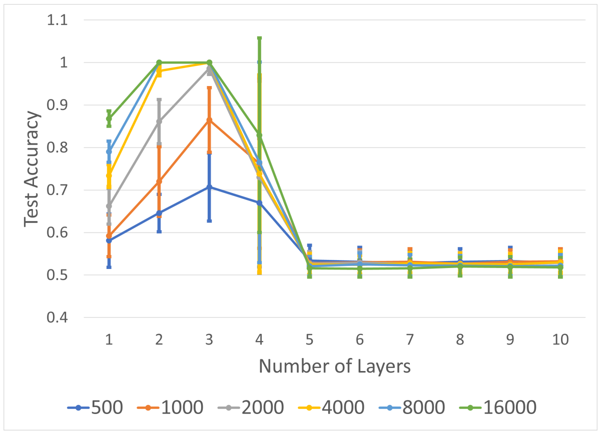
(a) $p_{\text{intraclass}}$



(b) $p_{\text{interclass}}$



(c) μ_{signal}

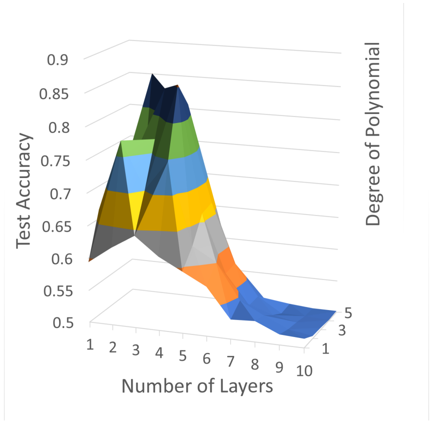


(d) Number of Nodes

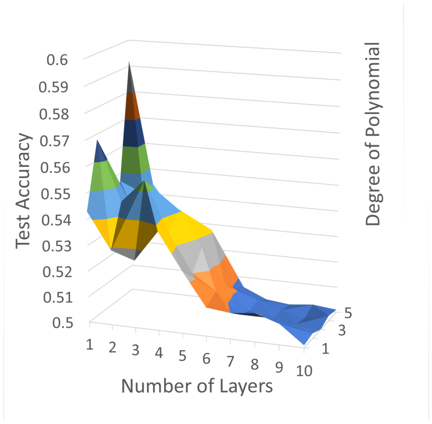
Figure 4.7: GCN: Test accuracy vs number of layers and different Erdős-Rényi model generation parameters. Subcaption indicates the model generation parameter that is changed for different colored lines. Colored horizontal and vertical lines represent average test accuracy and standard deviation, respectively.

intraclass edge creation probability is equal to the interclass edge creation probability, the network performs close to random since the graph is not useful. This is essentially using just the graph signal and the intraclass difference in means of graph signals is too small. At high intraclass edge creation probability, GCNs can achieve 100% accuracy and they resist better to the over-smoothing effect (accuracy drops slower at higher layers).

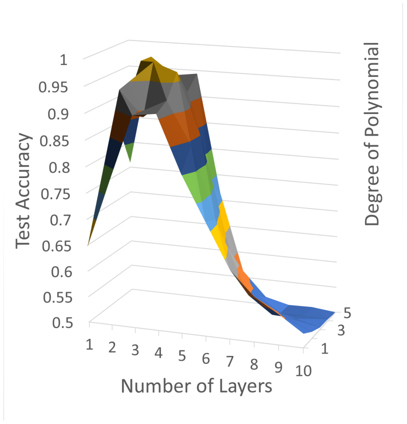
Fig. 4.8a, 4.8b, 4.8c show the results of TAGCN vs number of layers and degree of polynomial filters for 0.07, 0.05, and 0.09 intraclass edge creation probability, respectively. Like GCN, in general, as the intraclass edge creation probability increases, the test accuracy improves across both layers and degree of polynomial filters. The effect of the number of layers is similar to GCN, while the degree of polynomial filters seem



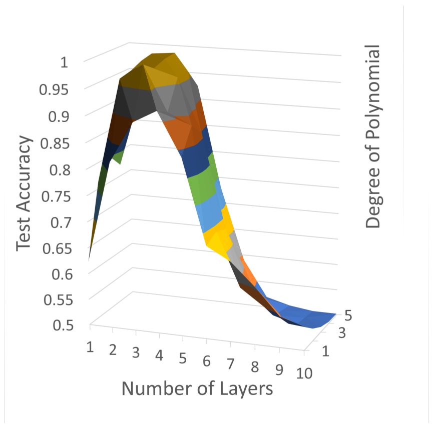
(a) Default ($p_{\text{intraclass}} = 0.07$, $p_{\text{interclass}} = 0.03$, $\mu_{\text{signal}} = \pm 0.1$, Number of Nodes $= 1000$)



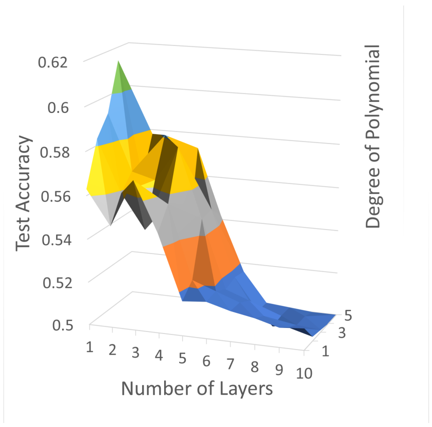
(b) $p_{\text{intraclass}} = 0.05$



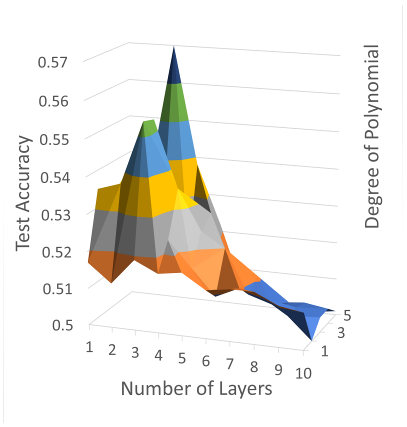
(c) $p_{\text{intraclass}} = 0.09$



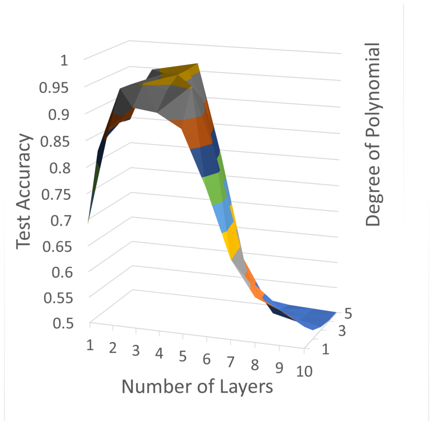
(d) $p_{\text{interclass}} = 0.02$



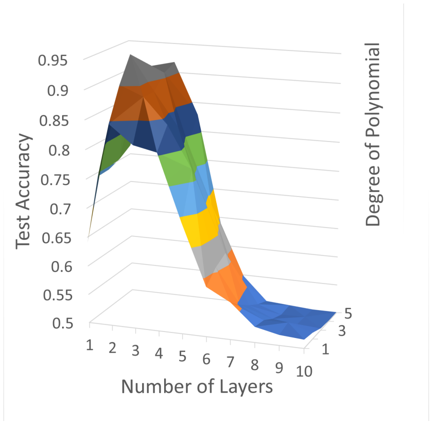
(e) $p_{\text{interclass}} = 0.04$



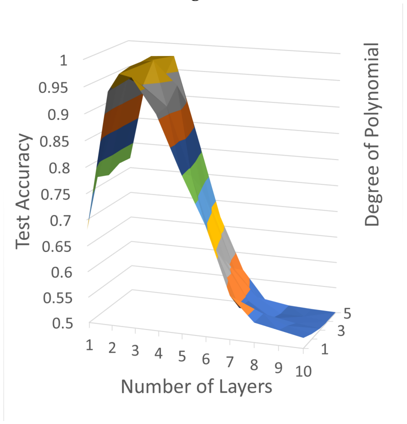
(f) $\mu_{\text{signal}} = 0$



(g) $\mu_{\text{signal}} = \pm 0.2$



(h) Number of Nodes $= 2000$



(i) Number of Nodes $= 4000$

Figure 4.8: TAGCN: Test accuracy vs degree of the polynomial filters, number of layers, and different Erdős-Rényi model generation parameters. Subcaption indicates the model generation parameter that is changed from the default setup.

to have a higher optimal value for lower layers and lower for higher layer, e.g., for the default (0.07) case for layers 1-10, the optimal degrees of the polynomial filters are 3, 5, 4, 3, 4, 2, 2, 1, 1, 1 .

Impact of Interclass Edge Creation Probability $p_{\mathbf{interclass}}$ Fig. 4.7b shows the results of GCN vs number of layers for different interclass edge creation probability. In general, as this probability increases, the accuracy drops, and the network over-smoothes at lower layer. This effect is opposite of the intraclass edge creation probability.

Fig. 4.8a, 4.8d, 4.8e show the results of TAGCN vs number of layers and degree of polynomial filters for 0.03, 0.02, and 0.04 interclass edge creation probability, respectively. Like GCN, in general, as the interclass edge creation probability increases, the test accuracy decreases across both layers and degree of polynomial filters. The effect of the number of layers is similar to GCN, while higher degree of polynomial filters seem to produce higher accuracy for lower layers.

Impact of Graph Signal Fig. 4.7c shows the results of GCN vs number of layers for different interclass difference in means of signals. As the difference increases, the accuracy improves as expected. What is notable is that it has little effect on the optimal number of layers and the over-smoothing phenomenon, implying that the relative accuracy vs number of layers is affected more by the graph structure than graph signal.

Fig. 4.8a, 4.8f, 4.8g show the results of TAGCN vs number of layers and degree of polynomial filters for 0.1, 0.0, and 0.2 interclass difference in means of signals, respectively. Like GCN, in general, as the interclass difference in means of signals increases, the test accuracy increases across both layers and degree of polynomial filters. It has little effect on the optimal number of layers, optimal degree of polynomial filters, and the over-smoothing phenomenon, implying that the relative accuracy vs number of layers and degree of polynomial is affected more by the graph structure than graph signal.

Impact of Number of Nodes Fig. 4.7d shows the results of GCN vs number of layers for different number of nodes. As the number of nodes increases, the accuracy improves. What is also interesting is that it has little effect on the optimal number of layers and the over-smoothing phenomenon, implying that the relative accuracy vs number of layers is not affected by the number of nodes if the other parameters are kept the same.

Fig. 4.8a, 4.8h, 4.8i show the results of TAGCN vs number of layers and degree of polynomial filters for 1000, 2000, 4000 number of nodes, respectively. Like GCN, in general, as the number of nodes increases, the accuracy improves across both layers and degree of polynomial filters. It also has little effect on the optimal number of layers, optimal degree of polynomial filters, and the over-smoothing phenomenon, implying that the relative accuracy vs the number of layers and degree of polynomial is not affected by the number of nodes if the other parameters are kept the same.

4.4.3.3 Smallworld

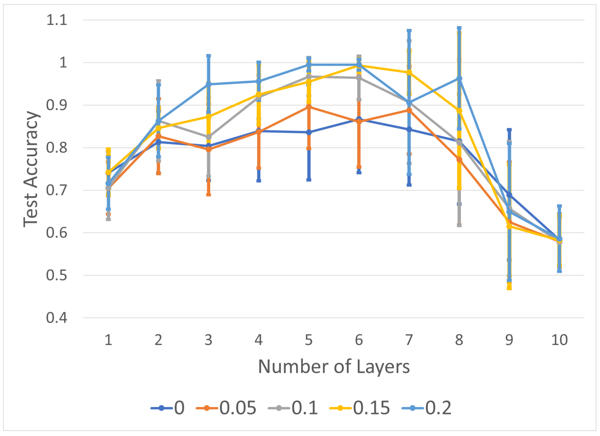
Fig. 4.9 and 4.10 show the results of GCN and TAGCN vs number of layers, number of layers + degree of the polynomial filters, respectively, for different smallworld graph characteristics generated using the Watts–Strogatz model. The graph characteristics are: mean degree, edge rewiring probability (to same class $p_{\text{intraclass}}$, and different class $p_{\text{interclass}}$), interclass difference in mean of graph signal, and number of nodes.

Impact of Intraclass Edge Rewiring Probability $p_{\text{intraclass}}$ Fig. 4.9a shows the results of GCN vs number of layers for different intraclass edge rewiring probability. In general, as the probability increases, the test accuracy increases. This suggests that less small-worldness characteristic is better for performance. GCNs excel with more global, long-range connections over a number of local short-range connections.

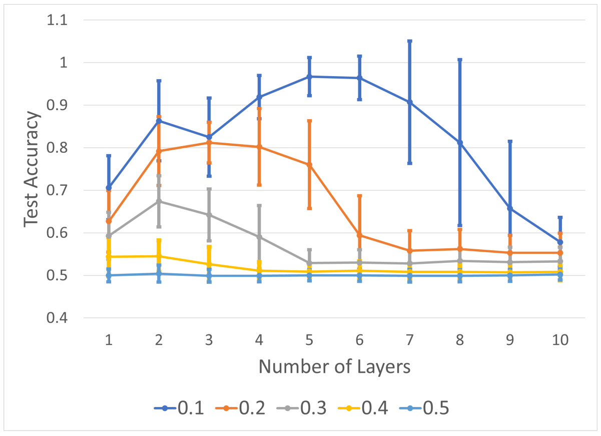
Fig. 4.10a, 4.10b, 4.10c show the results of TAGCN vs number of layers and degree of polynomial filters for 0.1, 0.0, and 0.2 intraclass edge rewiring probability, respectively. Like GCN, in general, as the intraclass edge rewiring probability increases, the test accuracy improves across both layers and degree of polynomial filters. The effect of the number of layers is similar to GCN. The filters still counter the effects of over-smoothing, with increased intraclass edge rewiring probability leading to higher number of layers before over-smoothing and sharper decline in accuracy when it occurs.

Impact of Interclass Edge Rewiring Probability $p_{\text{interclass}}$ Fig. 4.9b shows the results of GCN vs number of layers for different interclass edge rewiring probability. In general, as the probability increases, the test accuracy deteriorates as expected. When it reaches 0.5, the graph structure is not useful for classification, and the graph signals ($\mu_{\text{signal}} = \pm 0.1$ means) are not sufficiently distinguishable.

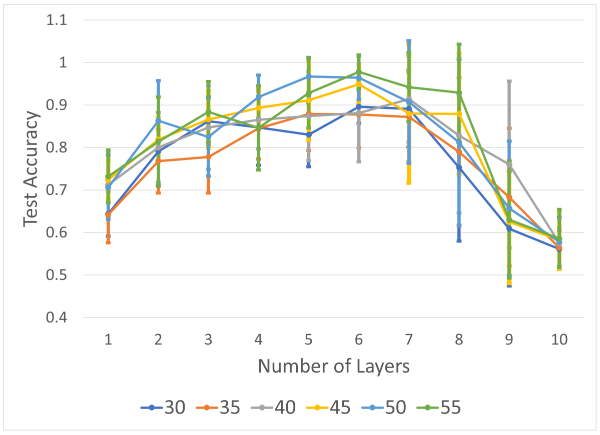
Fig. 4.10a, 4.10d, 4.10e show the results of TAGCN vs number of layers and degree of polynomial filters for 0.1, 0.2, and 0.3 interclass edge rewiring probability, respectively. Like GCN, in general, as the interclass edge rewiring probability increases, the



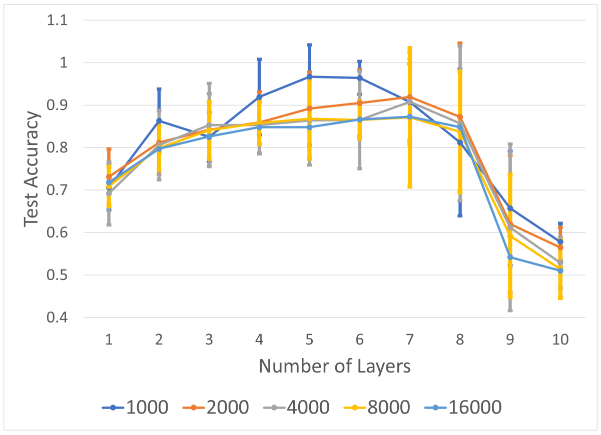
(a) $p_{\text{intraclass}}$



(b) $p_{\text{interclass}}$



(c) Mean degree κ



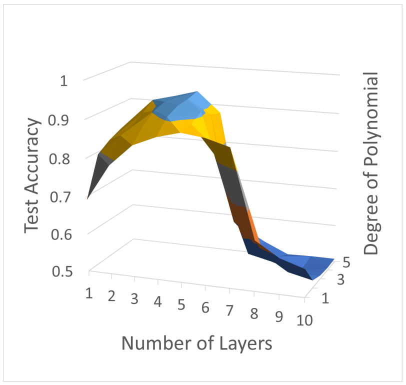
(d) Number of nodes

Figure 4.9: GCN: Test accuracy vs number of layers and different smallworld model generation parameters. Subcaption indicates the model generation parameter that is changed for different colored lines. Colored horizontal and vertical lines represent average test accuracy and standard deviation, respectively.

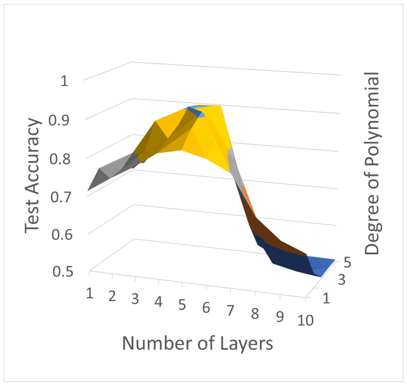
test accuracy decreases across both layers and degree of polynomial filters. The effect of the number of layers is similar to GCN. The filters no longer counter the effects of over-smoothing at high probability, with GNN performance behaving closer to that of Erdős-Rényi as the graph becomes more Erdős-Rényi-like.

Impact of Mean Degree κ Fig. 4.9c shows the results of GCN vs number of layers for different mean degree of graph. As the degree increases, the accuracy improves slightly across the first 9 layers, and higher number of layers leads to better performance.

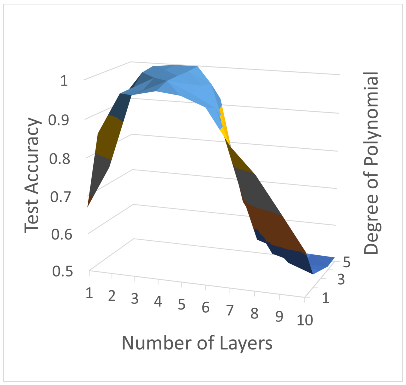
Fig. 4.10a, 4.10f, 4.10g show the results of TAGCN vs number of layers and degree of polynomial filters for 50, 30, 40 mean degree, respectively. As the mean degree increases,



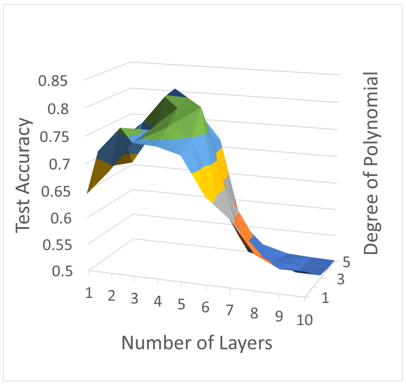
(a) Default ($p_{intraclass} = 0.1$, $p_{interclass} = 0.1$, $\kappa = 50$, $\mu_{\text{signal}} = \pm 0.1$, number of nodes $= 1000$)



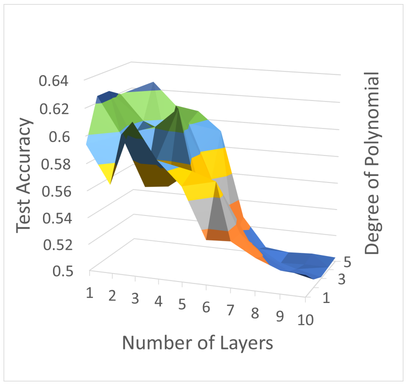
(b) $p_{intraclass} = 0.0$



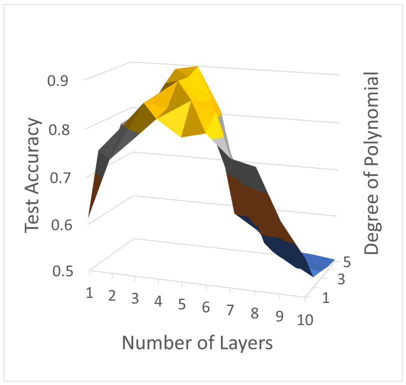
(c) $p_{intraclass} = 0.2$



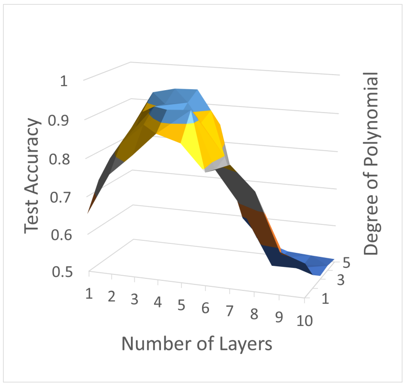
(d) $p_{interclass} = 0.2$



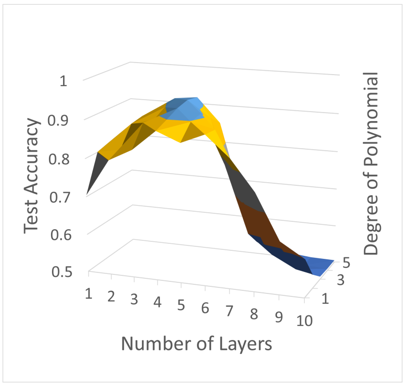
(e) $p_{interclass} = 0.3$



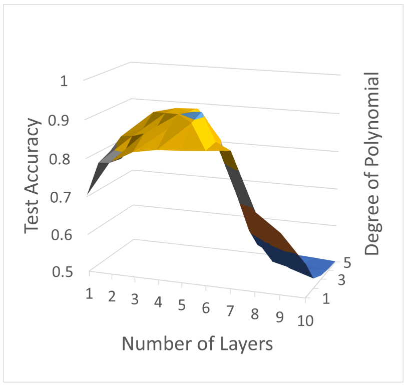
(f) $\kappa = 30$



(g) $\kappa = 40$



(h) number of nodes $= 2000$



(i) number of nodes $= 4000$

Figure 4.10: TAGCN: Test accuracy vs number of layers, degree of the polynomail filters, and different smallworld model generation parameters. Subcaption indicates the model generation parameter that is changed from the default setup.

the accuracy increases for different number of layers and degree of polynomial filters. The optimal number of layers stays about the same, and higher mean degree seems to lead to more rapid decline in accuracy from the optimum.

Impact of Number of Nodes Fig. 4.9d shows the results of GCN vs number of layers for different number of nodes. Similar to its effect on Erdős-Rényi, the accuracy increases as the number of nodes increases, reaching an optimum at around 3 layers. The number of nodes also has little effect on the optimal number of layers and the over-smoothing phenomenon, implying that the relative accuracy vs number of layers is not affected by the number of nodes if the other parameters are kept the same.

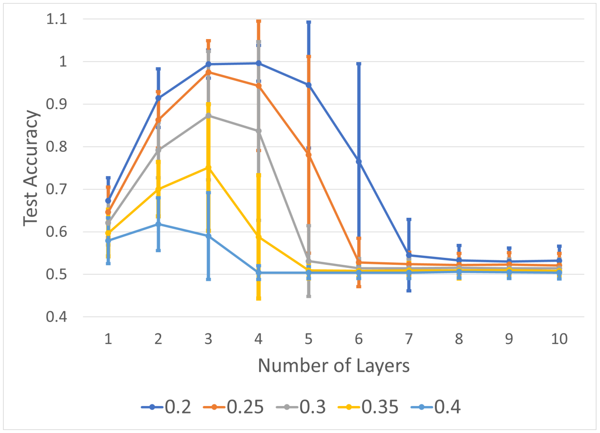
Fig. 4.10a, 4.10h, 4.10i show the results of TAGCN vs number of layers and degree of polynomial filters for 1000, 2000, 4000 number of nodes, respectively. Unlike GCN, in general, as the number of nodes increases, the accuracy stays about the same. The number of nodes also has little effect on the optimal number of layers, optimal degree of polynomial filters, and the over-smoothing phenomenon, implying that the relative accuracy vs the number of layers and degree of polynomial is not affected by the number of nodes if the other parameters are kept the same.

4.4.3.4 Preferential Attachment

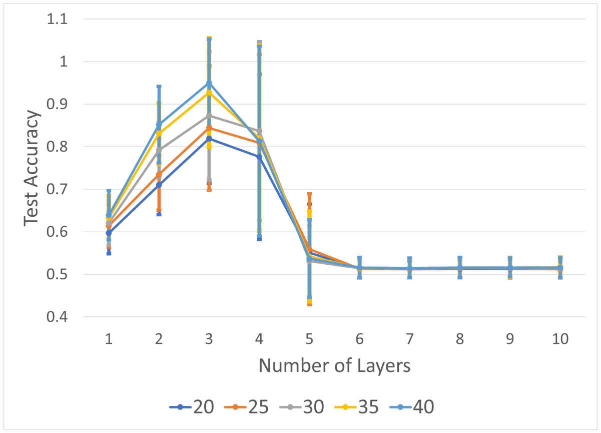
Fig. 4.11 and 4.12 show the results of GCN and TAGCN vs number of layers, number of layers + degree of the polynomial filters, respectively, for different preferential attachment graph characteristics generated using the Barabási-Albert model. The graph characteristics are: edge rewiring probability (to different class $p_{\text{interclass}}$), edges-to-attach m, interclass difference in mean of graph signal, and number of nodes.

Impact of Interclass Edge Rewiring Probability $p_{\text{interclass}}$ Fig. 4.11a shows the results of GCN vs number of layers for different interclass edge rewiring probability. In general, as this probability increases, the accuracy drops, and the network over-smoothes at lower layer. The effect is very similar to the interclass edge creation probability of Erdős-Rényi (see Fig. 4.7b).

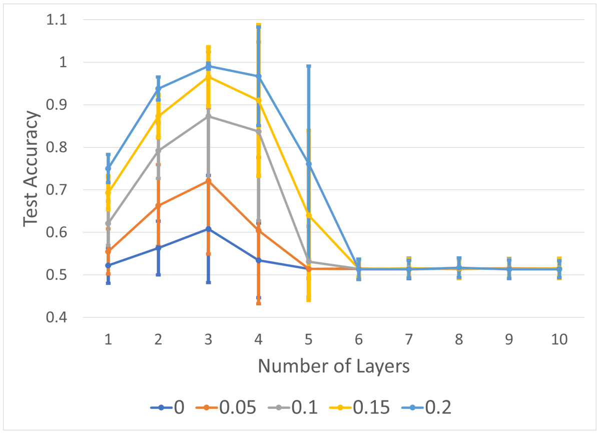
Fig. 4.12a, 4.12b, 4.12c show the results of TAGCN vs number of layers and degree of polynomial filters for 0.3, 0.25, and 0.35 interclass edge rewiring probability, respectively. Like GCN, in general, as the probability increases, the test accuracy decreases across both layers and degree of polynomial filters. The effect is again very similar to the interclass



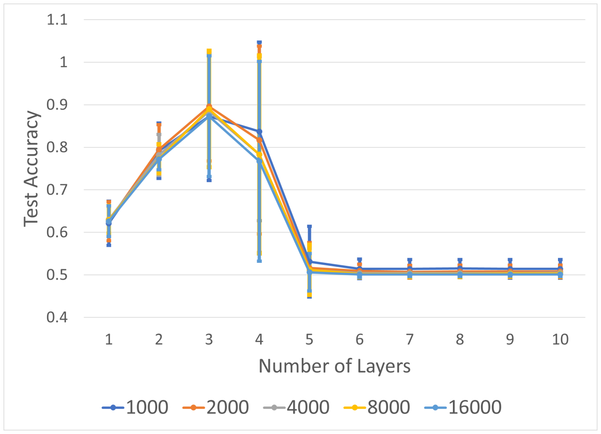
(a) $p_{\text{interclass}}$



(b) Edges-to-attach m



(c) μ_{signal}



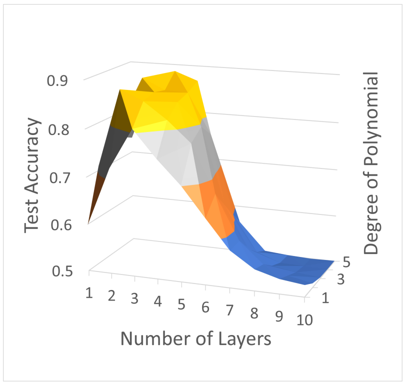
(d) Number of nodes

Figure 4.11: GCN: Test accuracy vs number of layers and different preferential attachment model generation parameters. Subcaption indicates the model generation parameter that is changed for different colored lines. Colored horizontal and vertical lines represent average test accuracy and standard deviation, respectively.

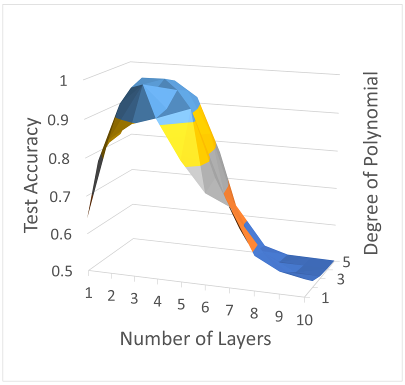
edge creation probability of Erdős-Rényi (see Fig. 4.8d). Higher degree of the polynomial filters seems to produce higher accuracy for lower layers, and lower accuracy for higher layers.

Impact of Edges-to-attach m Fig. 4.11b shows the results of GCN vs number of layers for different edges-to-attach m. As the number of edges-to-attach increases, the accuracy improves as expected. It has little effect on the optimal number of layers and the over-smoothing phenomenon, implying that the relative accuracy vs number of layers is not affected by this parameter.

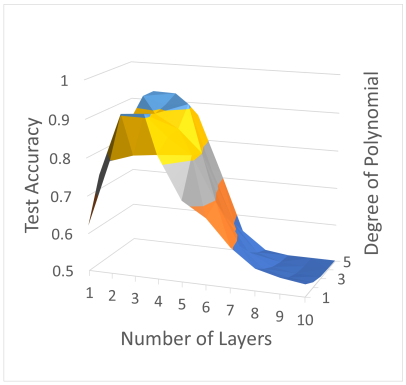
Fig. 4.12a, 4.12d, 4.12e show the results of TAGCN vs number of layers and degree of



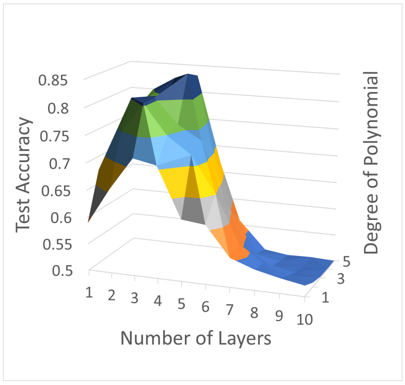
(a) Default ($p_{\text{interclass}} = 0.3$, edges-to-attach $m = 30$, $\mu_{\text{signal}} = \pm 0.1$, Number of nodes $= 2000$)



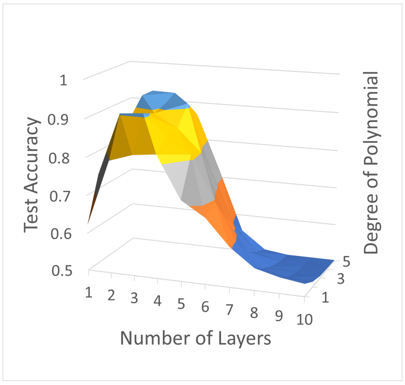
(b) $p_{\text{interclass}} = 0.25$



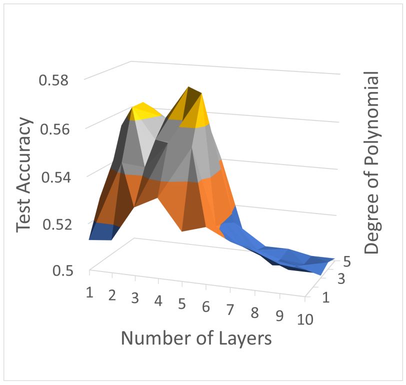
(c) $p_{\text{interclass}} = 0.35$



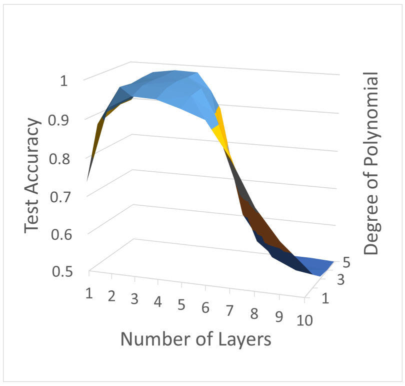
(d) edges-to-attach $m = 20$



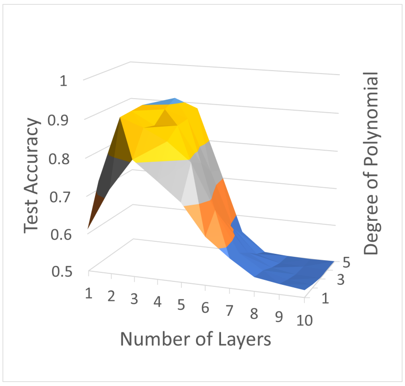
(e) $m = 40$



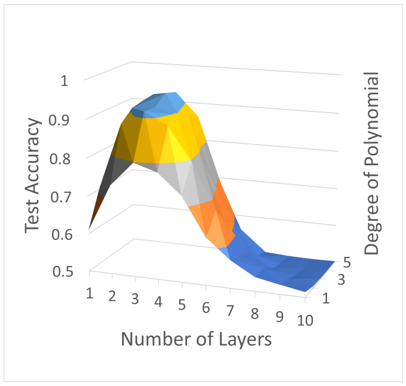
(f) $\mu_{\text{signal}} = 0$



(g) $\mu_{\text{signal}} = \pm 0.2$



(h) Number of nodes $= 2000$



(i) Number of nodes $= 4000$

Figure 4.12: TAGCN: Test accuracy vs number of layers, degree of the polynomail filters, and different preferential attachment model generation parameters. Subcaption indicates the model generation parameter that is changed from the default setup.

polynomial filters for 30, 20, and 40 edges-to-attach, respectively. Like GCN, in general, as the number increases, the test accuracy increases across both layers and degree of polynomial filters.

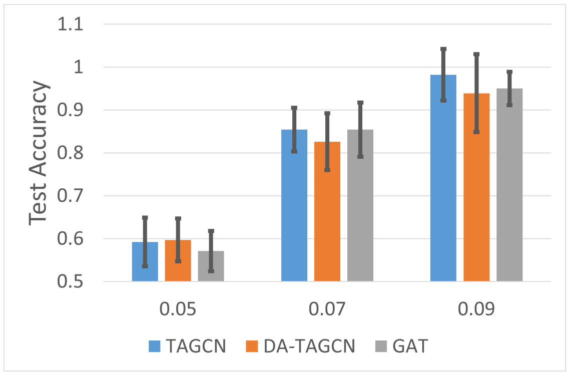
Impact of Graph Signal The behavior relative to the interclass difference between graph signals is similar to that of Erdős-Renŷi. Fig. 4.11c shows the results of GCN vs number of layers for different interclass difference in means of signals. As the difference increases, the accuracy improves as expected. Fig. 4.12a, 4.12f, 4.12g show the results of TAGCN vs number of layers and degree of polynomial filters for 0.1, 0.0, and 0.2 interclass difference in means of signals, respectively. Like GCN, in general, as the interclass difference in means of signals increases, the test accuracy increases across both layers and degree of polynomial filters. It has little effect on the optimal number of layers, optimal degree of polynomial filters, and the over-smoothing phenomenon, implying that the relative accuracy vs number of layers and degree of polynomial is affected more by the graph structure than graph signal.

Impact of Number of Nodes Fig. 4.11d shows the results of GCN vs number of layers for different number of nodes. As the number of nodes increases, the accuracy remains unchanged (whereas for Erdős-Renŷi, it increases, and plateau at 100%).

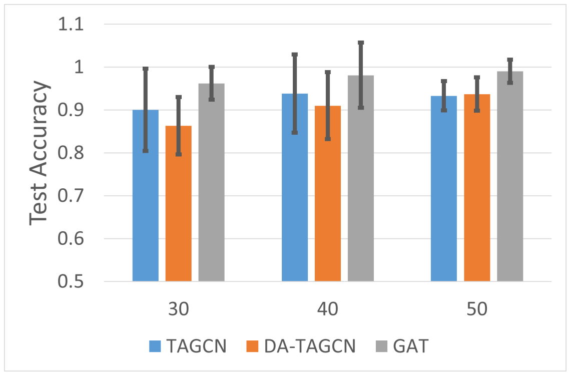
Fig. 4.12a, 4.12h, 4.12i show the results of TAGCN vs number of layers and degree of polynomial filters for 1000, 2000, 4000 number of nodes, respectively. Like GCN, in general, as the number of nodes increases, the accuracy remains similar across both layers and degree of polynomial filters. It also has little effect on the optimal number of layers, optimal degree of polynomial filters, and the over-smoothing phenomenon, implying that the accuracy vs the number of layers and degree of polynomial is not affected by the number of nodes if the other parameters are kept the same.

4.4.3.5 GNN Variants

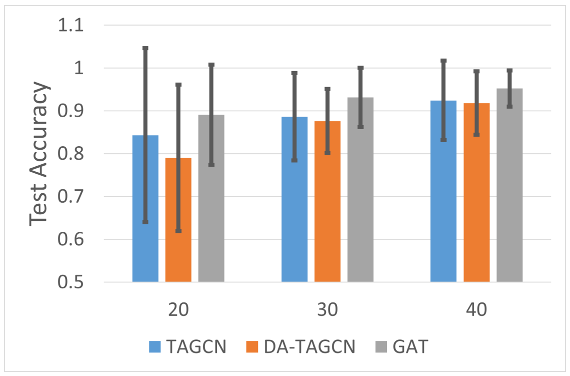
We consider two GNN variants: degree-aware TAGCNs (DA-TAGCNs), and graph attention networks (GATs) [50]. See Section 4.3.3.3 for their descriptions. Fig. 4.13 shows their results on the 3 models of synthetic datasets. DA-TAGCNs perform worse or similar to TAGCN for all datasets. This is because in general, there is no significant differences in number of intraclass edges for nodes of different classes. GAT performs similarly for different intraclass edge creation probability for Erdős-Rényi datasets while consistently



(a) Erdős-Rényi Intraclass Edge Creation Probability $p_{\text{intraclass}}$



(b) Smallworld mean degree κ



(c) Preferential Attachment Edges-to-attach m

Figure 4.13: TAGCN vs GNN Variants (DA-TAGCN vs GAT) for 3 synthetic graph generation models. Subcaption indicates the model generation parameter plotted on the x-axis. Colored bars represent average test accuracy of each architecture. Vertical black lines on the bars represent standard deviation.

perform better for different mean degree of smallworld models and different edges-to-attach for preferential attachment models. This is also expected since the edge weights were originally one, and hence can be improved using the signals.

4.5 Conclusions

In conclusion, we have shown how the graph structure affects the performance of GNNs for the node classification task on real and synthetic data. In general, for the two main GNN architectures that we studied (GCN and TAGCN), not many layers are needed to achieve optimal performance (compared with computer vision and natural language processing). TAGCN requires fewer number of layers than GCN, with low degree of the polynomial filters. Unlike graph classification, studied in Chapter 5, graph signal is necessary and important. For some datasets, classifying using a simple estimator on the graph signals can outperform GNNs. For synthetic datasets, both GCNs and TAGCN perform similarly on Erdős-Rényi and preferential attachment graphs with respect to the number of layers. For smallworld graphs, TAGCN filters play an important role in achieving the optimal accuracy and accelerating the effect of over-smoothing.

For real datasets, node classification significantly improved when the graph is taken into consideration. We relate simple metrics on real datasets to the performance of these models. We show, for datasets with higher difference in % of intraclass and interclass edges like Coauthor Physics, the effect of over-smoothing due to number of layers is smaller. We also show that over-smoothing occurs slower vs the degree of polynomial filters for datasets with low average degree. Since the node degrees are in general common among different classes, DA-GCN and DA-TAGCN do not perform better than GCN and TAGCN, respectively. Graph attention network only performs well on two datasets, suggesting that only those two have graph signals useful for learning edge weights.

For synthetic datasets, we relate the performance to different graph characteristics. We find that, in general, high intraclass edge ratio not only leads GNN achieving higher accuracy, but also leads to them resisting over-smoothing more for both number of layers and degree of polynomial filters (which has higher optimal values for lower layers, and lower optimal values for higher layers). Changing the graph signals by increasing the interclass difference between the means of the graph signal improves the accuracy while not affecting the optimal number of layers and degree of polynomial filters. For both architectures, increasing the number of nodes for Erdős-Rényi and preferential attachment models improves the test accuracy but for preferential attachment it only lowers the variance of the test accuracy if all other graph characteristics remain the same. Just like for real datasets, DA-TAGCN performs worse than TAGCN. GAT in general performs better because the graph signals are distinct between classes and the edge weights are the same.

Chapter 5: Graph Classification

Graph classification is a task in geometric deep learning with many practical applications. In graph classification, we infer graph-level properties or classes of unlabeled graphs, given a set of labeled graphs with graph structures and signals. This is analogous to the image classification task studied in computer vision. For example, in MUTAG (see Fig. 5.1), molecules are graphs defined according to the chemical bonds between atoms and the task is to predict whether a molecule is mutagenic or not. Geometric deep learning methods have achieved state-of-the-art over this and many graph classification problems [74, 21, 22].

In this chapter, we take a closer look as to why they perform well, by analyzing how data graph structure[1] affects the performance of these methods using real and synthetic datasets. For real datasets, we look at how simple metrics on real datasets can give clues to the performance of these methods and insights into selecting the appropriate

[1]We refer to "data graph structure" as "graph structure," and "data graph signal" as "graph signal" or "node feature."

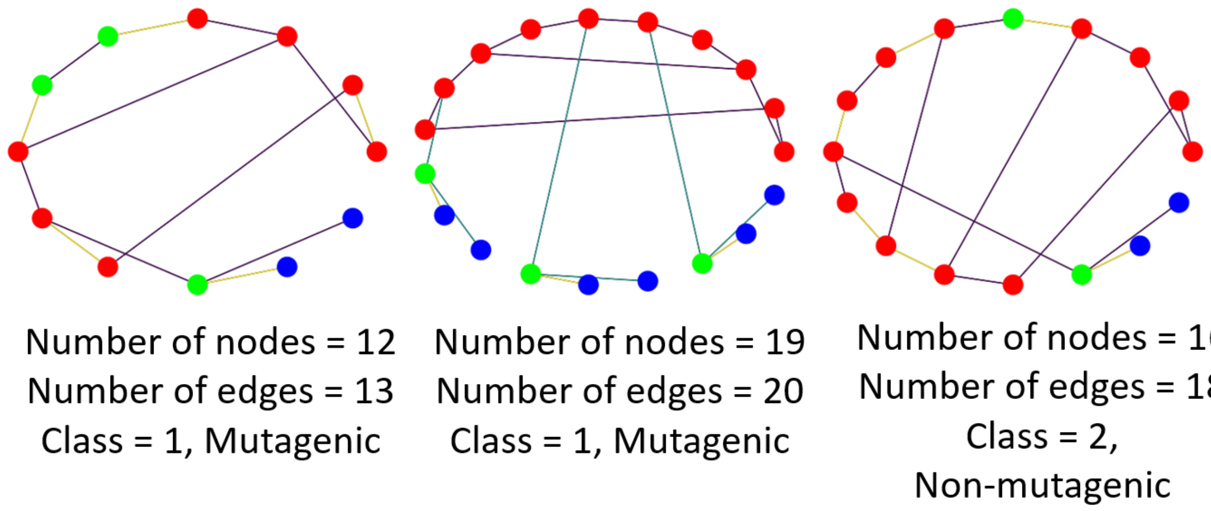


Figure 5.1: MUTAG dataset [1]: 188 graphs, 2 classes.

architectures. The results motivate a more comprehensive investigation using synthetic datasets where we generate many graphs and analyze several model generation parameters/graph characteristics, e.g., edges-to-attach of preferential attachment model.

We organize the chapter as follows. We begin with the task definition for graph classification (Section 5.1). Next, we provide some background on some of the issues in graph classification that we are addressing (Section 5.2), like limit on the expressive power of GNNs (Section 5.2.4) and impact of GNN architecture (Section 5.2.3). Then we break our analysis into two parts. First, we study how simple metrics on real datasets can give clues to the performance of these methods and insights into selecting the appropriate deep model architectures (Section 5.3). The results motivate a more comprehensive investigation using synthetic datasets with many different data structures (Section 5.4). Finally, we summarize the results in the conclusions (Section 5.5).

5.1 Task Definition

In graph classification, we try to discriminate between graphs of different classes. This is generally a supervised task where a subset of graphs are labeled and used in training. It can be unsupervised by not using any label and by learning entirely through clustering graph features/statistics. The labels for the graphs can be categorical valued (binary or multiclass classification), or continuous valued (regression).

We now give a more formal definition following the notation defined in Section 2.1.1. We consider the supervised learning setting with the following observations in the forms of graphs $\mathcal{G}_s = (\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_N)$, and labels $\mathbf{Y}_s = (\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_N)$. We are given a subset of labels for training $\{\mathbf{Y}_i \mid i \in D_{\text{training}}\}$, and the task is to predict the rest $\{\mathbf{Y}_i \mid i \in D_{\text{testing}}\}$, where D_{training} and D_{testing} correspond to the train and test sets, respectively.

Each graph $\mathcal{G}_i$ consists of a set of nodes V_i with $|V_i| = N_i$, signal $\mathbf{X}_i \in \mathbb{R}^{N_i \times C_0}$ on the nodes (where C_0 is the number of features on each node), a set of edges $\mathcal{E}_i$, and an adjacency matrix $\mathbf{A}_i$. $[\mathbf{A}_i]_{jk} = 0$ unless there is an edge $e = (j, k)$ connecting node j and node k.

Given a neural network function $f(\mathbf{X}_i, \mathbf{A}_i)$, we use a non-negative real-valued loss function to model the prediction error of each sample:

$$L(f(\mathbf{X}_i, \mathbf{A}_i), \mathbf{Y}_i) \tag{5.1}$$

Now we can define the true risk associated with the function f by interpreting data as independent and identically distributed (i.i.d) random variables:

$$R_{\text{true}}(f) = \mathbb{E}_{\mathbf{X},\mathbf{A}\sim P}(L(f(\mathbf{X},\mathbf{A}),\mathbf{Y})) = \int L(f(\mathbf{X},\mathbf{A}),\mathbf{Y})dP(\mathbf{X},\mathbf{A},\mathbf{Y}) \tag{5.2}$$

where P is the true distribution over the inputs. In practice, however, we do not have the true distribution P, so instead we use the empirical risk by averaging the loss function on the training set, i.e.,:

$$R_{\text{empirical}}(f) = \frac{1}{N_{\text{training}}} \sum_{i\in D_{\text{training}}} L(f(\mathbf{X}_i,\mathbf{A}_i),\mathbf{Y}_i). \tag{5.3}$$

where $N_{\text{training}} = |D_{\text{training}}|$. The concept of choosing the function f^* that minimizes this empirical risk is the empirical risk minimization:

$$f^* = \underset{f\in F}{\operatorname{argmin}}\, R_{\text{empirical}}(f) \tag{5.4}$$

where F is a fixed class of functions/neural networks under consideration. For GCNs and TAGCNs, this refers to the set of weights $\mathbf{W}^\star = (\mathbf{W}^\mathbf{0},\mathbf{W}^\mathbf{1},\ldots,\mathbf{W}^\mathbf{L})$ and $\mathbf{W}^\star = (\mathbf{W}_\mathbf{0}^\mathbf{0},\ldots\mathbf{W}_\mathbf{K}^\mathbf{0},\mathbf{W}_\mathbf{0}^\mathbf{1},\ldots,\mathbf{W}_\mathbf{K}^\mathbf{1},\ldots\mathbf{W}_\mathbf{K}^\mathbf{L})$ respectively, as seen in (3.1) and (3.2), copied here. For GCN:

$$\mathbf{X}^{(\ell+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}^{(\ell)}\mathbf{W}^{(\ell)}\right), \tag{3.1}$$

and for TAGCN:

$$\mathbf{X}^{(\ell+1)} = \sigma\left(\sum_{k=0}^{K}\mathbf{A}^k\mathbf{X}^{(\ell)}\mathbf{W}_k^{(\ell)}\right). \tag{3.2}$$

We denote $\widehat{Y} = f(\mathbf{X},\mathbf{A})$ as the output of the neural network. For graph classification, this can be for example a 2-layer TAGCN, followed by 2 a few MLPs and aggregation layer (see Section 3.4). The training loss function is thus:

$$L(\widehat{\mathbf{Y}}_i,,\mathbf{Y}_i) \tag{5.5}$$

where $\widehat{\mathbf{Y}}_i$ corresponds to the logits predicted by the neural network during training (which argmax corresponds to the predicted graph labels). For multi-class classification, we use softmax cross-entropy loss shown:

$$L(\widehat{Y}_i,,Y_i) = -\sum_{i\in N_c}\log(S(\widehat{\mathbf{Y}}_i))(\mathbf{Y}_v)_i \tag{5.6}$$

where N_C is the number of classes and $(Y_v)_i$ is the one-hot encoding of the class of the true label (all 0 except for 1 for each label). The softmax function $S(\cdot)$ used to convert the logits to probabilities is shown in (4.7), copied here:

$$S((\widehat{\mathbf{Y}}_v)_i) = \frac{\exp((\widehat{\mathbf{Y}}_v)_i)}{\sum_{j=1}^{N_c} \exp((\widehat{\mathbf{Y}}_v)_j)}. \tag{4.7}$$

We use the above metrics during training and validation. For testing, we report the performance based on the test accuracy (using argmax to select the predicted class $\widehat{Y}_{\text{testing}}$).

5.2 Issues Considered

In this section, we present the main issues for the graph classification task that we consider using real and synthetic datasets.

5.2.1 Impact of Graph Structure

Just like node classification, the primary challenge that we are addressing is how the data graph structure affects the performance of graph neural networks. Unlike node classification, we are considering multiple graph structures of different classes.

For real datasets, we compare the test accuracy of classification using graph NNs vs using simpler machine learning algorithms on network statistics and metrics. For synthetic datasets, we investigate when the graph structures are distinct between classes, based on the graph generation parameters, e.g., edge creation probability for different classes for Erdős-Rènyi graphs.

5.2.2 Impact of Graph Signal

Graph signals play a very important role. There are cases when just using the simple statistics like average and variance on the signals is enough for good classification performance. Signals can also work against the graph structure, by being indistinguishable between classes or by being opposite in an XOR fashion as shown in Table 5.1.

Table 5.1: XOR Case of Graph Signal vs Graph Structure

	Graph Structure Type A	Graph Structure Type B
Signal Type A	Class 0	Class 1
Signal Type B	Class 1	Class 0

5.2.3 Impact of Graph Neural Network Architecture

Alongside our above objective, we also study how the GNN architecture affects its performance. We consider two methods: graph convolution networks (GCNs) [17] and topology adaptive graph neural networks (TAGCNs) [48]. We look into hyperparameters like the number of layers for GCN and TAGCN (defined in 3.2). For the latter, we also look into the degree of the polynomial filters.

For both real and synthetic datasets, we study the architecture in conjunction with the graph structure. We also consider simple variants of these methods that can better capture some of the real graph characteristics and synthetic graph generation parameters, hence improving performance.

5.2.4 Limit on the Expressive Power of GNNs

While performing well on many datasets including citation networks [48] and NYC taxi pickup data [75], GNNs have been theoretically shown to fail on certain types of graph structures and signals [76], described below.

Consider two graphs with adjacency matrices $\mathbf{A}_1$, $\mathbf{A}_2$ respectively. Let the graph signals be $\mathbf{X}_1$ and $\mathbf{X}_2$ respectively. Polynomial filter TAGCN fails when $P(\mathbf{A}_1)\mathbf{X}_1 = P(\mathbf{A}_2)\mathbf{X}_2$ (where P is a polynomial filter), but $\mathbf{A}_1$ and $\mathbf{A}_2$ are not isomorphic, i.e., $\mathbf{A}_1 \neq \mathbf{P}\mathbf{A}_2\mathbf{P}^{-1}$ for any permutation matrix $\mathbf{P}$. For example, consider the graphs in Fig. 5.2 [76].

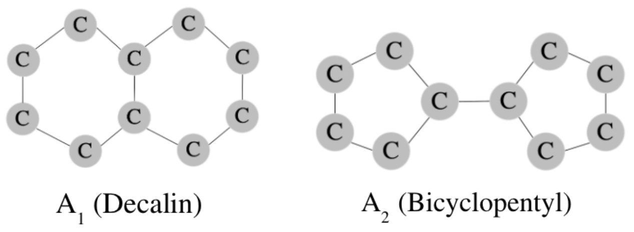


Figure 5.2: Two non-isomorphic graphs where $\mathbf{A}_1 \neq \mathbf{A}_2$. The graph signal is $\mathbf{X}_1 = \mathbf{X}_2 = [6, 6, 6, 6, 6, 6, 6, 6, 6, 6]^T$, where the number 6 represents the element Carbon at each node of the graph.

In the graphs in Fig. 5.2, the graphs are not isomorphic, $\mathbf{A}_1 \neq \mathbf{P}\mathbf{A}_2\mathbf{P}^{-1}$, but every node is connected to the same valued neighbors. The two chemicals shown, Decalin and Bicyclopentyl, are represented by graphs where the nodes are the Carbon atoms and the edges are the chemical bonds between the atoms. The graph signal is $\mathbf{X}_1 = \mathbf{X}_2 = [6, 6, \ldots, 6]^T$ where the number 6 represents the element Carbon at each node in the graph[2]. Because of this, since $\mathbf{A}x$ is a weighted sum of the neighbors of each node,

$$\mathbf{A}_1\mathbf{X}_1 = \mathbf{A}_2\mathbf{X}_2 = [18, 18, 12, 12, 12, 12, 12, 12, 12, 12]^T \tag{5.7}$$

In addition, $\mathbf{P}(\mathbf{A}_1)\mathbf{X}_1 = \mathbf{P}(\mathbf{A}_2)\mathbf{X}_2$. Thus, $\sigma(\mathbf{P}(\mathbf{A}_1)\mathbf{X}_1) = \sigma(\mathbf{P}(\mathbf{A}_2)\mathbf{X}_2)$. Because of this, TAGCN (and GCN) convolution produces the same output and thus it cannot distinguish between $\mathbf{A}_1$ and $\mathbf{A}_2$.

This phenomenon can occur in chemical datasets such as MUTAG [1]. Many chemicals have a similar structure and contain the same elements and, thus, are more likely to fail with GNNs [76].

These failure cases can be tied to the Weisfeiler-Lehman (WL) test of isomorphism on graphs [77]. Reference [24] shows that GNNs are at most as powerful as the WL graph isomorphism test. In other words, if this test cannot distinguish between two nonidentical graphs, then GNNs would not be able to distinguish between them. For a more theoretical examination, see also [25, 26].

There are several ways to improve the expressive power of GNNs and to deal with these peculiar cases. We propose a solution using a spectral domain shift M based on concepts from GSP (see Section 2.3). This solves the problem of GNNs failing when applied to certain graph inputs [78].

As described in [78], using the example in Fig. 5.2, we use (3.7) to obtain the $\mathbf{M}_1$ and $\mathbf{M}_2$, the spectral graph shifts corresponding to $\mathbf{A}_1$ and $\mathbf{A}_2$, respectively. These are shown in Fig. 5.3.

In general, this is not an issue for real datasets, as almost all the datasets that we encounter do not have non-isomorphic but similar graphs and signals like the ones shown in Fig. 5.2.

Using 1-degree polynomial filter, $P(\mathbf{M}) = \mathbf{M}$, we obtain

$$\mathbf{M}_1\widehat{\mathbf{X}_1} = \\ [6.7, 0.8, -13.1, -3.4, -3, 1.7, -12.8, -7, 17.9, -4.3]^T$$

[2]In this paper, we consider $\mathbf{X}_1 = \mathbf{X}_2 = [6, 6, \ldots, 6]^T$. In general, any vector $\mathbf{X}_1 = \mathbf{X}_2 = [a, a, \ldots, a]^T$, $a \in \mathbb{C}$ would also fail.

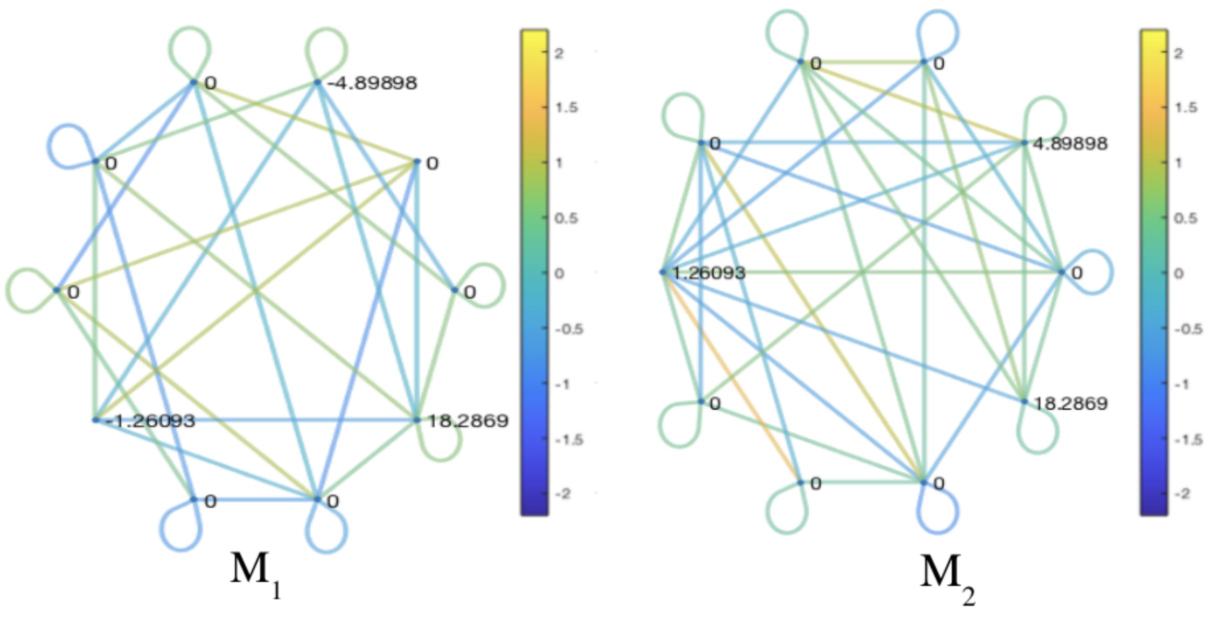


Figure 5.3: $\mathbf{M}_1$ and $\mathbf{M}_2$, the spectral graph shifts corresponding to $\mathbf{A}_1$ and $\mathbf{A}_2$ from Fig. 5.2, respectively. Each node is labeled with the values of $\widehat{\mathbf{X}_1}$ and $\widehat{\mathbf{X}_2}$, the GFT of $\mathbf{X}_1 = \mathbf{X}_2 = [6, 6, \ldots, 6]^T$ (values of carbon). The colors represent the edge weights of each edge in the graph.

$$\mathbf{M}_2\widehat{\mathbf{X}_2} = \\ [-3.2, -0.7, -0.6, 10.5, 0.8, 9.9, -10, 13.6, 16.4, -4.3]^T$$

Thus, $\mathbf{M}_1\widehat{\mathbf{X}_1} \neq \mathbf{M}_2\widehat{\mathbf{X}_2}$ and the spectral domain convolutional layer will produce different outputs.

5.3 Real Data

In this section, we analyze the impact of graph structure on the performance of GNNs for 11 real graph classification datasets. We divide this section into 3 parts: datasets description and their statistics, experimental setup, and results.

5.3.1 Data Description and Statistics

For real datasets, we study common graph classification benchmarks in biology and social networks. Their data statistics and network metrics are shown in tables 5.2 and

5.3. For averages, we take the mean across all the graphs. For diameter, we consider disjoint components separately and average them before averaging across all the graphs.

Table 5.2: Biological Networks Data Statistics [1]

Dataset	Size	Classes	Avg. Nodes	Avg. Degree	Avg. Diameter
MUTAG	188	2	17.9	2.2	8.2
ENZYMES	600	6	32.6	3.9	10.7
PROTEINS	1113	2	35.9	3.7	11.3
DD	1178	2	273.3	5.0	19.7
AIDS	2000	2	15.4	2.0	7.1
NCI1	4110	2	29.9	2.2	12.5

Table 5.3: Social Networks Data Statistics [1]

Dataset	Size	Classes	Avg. Nodes	Avg. Degree	Avg. Diameter
IMDB-Binary	1000	2	19.7	8.9	1.9
Reddit-Binary	2000	2	233.4	2.3	6.8
COLLAB	5000	3	72.6	36.7	1.9
Github_stargazers	12725	2	90.5	3.1	5.8
Twitch_egos	127094	2	29.7	5.4	2.0

5.3.1.1 Biological Network Data Descriptions

We describe the biological network data. Typically, each sample is a chemical, represented by a graph.

MUTAG [79] is a graph classification dataset where each graph is a molecule. Each node is an item, and the graph for each data point is formed according to the bonds between atoms. The task is to predict whether the molecule is mutagenic.

ENZYMES [80], PROTEINS [81], and DD [77] are graph classification datasets where each graph is a protein molecule. For ENZYMES and PROTEINS, nodes represent secondary structure elements (alpha helix, the beta-sheet, and the turn), and edges connect nodes that are along an amino acid sequence. For DD, each node represents individual amino acids, and edges are formed based on their spatial distance. The task is to predict whether the protein is an enzyme.

AIDS [82] is a graph classification dataset where each graph represents a compound. Each node is an atom, and edges represent chemical bonds. The task is to evaluate evidence of anti-HIV activity.

NCl1 [83] is a graph classification dataset where each graph is a chemical compound. Each node is an atom, and edges represent chemical bonds. The task is to predict the anti-cancer potential of each molecule.

5.3.1.2 Social Network Data Descriptions

We describe the social network datasets.

IMDB-Binary [84] is a graph classification dataset where each graph is an ego-network. Each node represents an actor/actress, and the edges are formed between those who appeared together in a movie. The task is to determine whether a graph comes from romance or action genre.

Reddit-Binary [84] is a graph classification dataset where each graph is a Reddit discussion thread/post. Each node represents a user from the online community. The edges are formed when one user comments on the other. The task is to predict which post of the online community this graph comes from.

COLLAB [84] is a graph classification dataset where each graph is a collaboration network. Each node represents a researcher in a certain field and collaboration between researchers form the edges. The task is to predict whether a graph belongs to one of three specific fields.

Github_stargazers [85] is a graph classification dataset where each graph represents a social network of GitHub users. Nodes are users and edges are their follower relationship. The task is to predict whether a graph belongs to web or machine learning developers.

Twitch_egos [85] is a graph classification dataset where each graph is a ego-net of Twitch users. Nodes are users and edges are friendships. The task is to predict whether the ego user plays a single or multiple-player games. Players who play a single game usually have a more dense ego-net.

5.3.2 Experimental Setup

For the neural network training, we perform stratified 10-fold cross-validation and report the final test accuracy optimized over all hyperparameters. The hyperparameters are the number of graph convolutional layers, the number of channels in each layer, dropout rates, and (for TAGCN) the degree of the polynomial filter. We use cross-entropy loss and ADAM optimizer with a starting learning rate of 0.01. Experiments are performed in PyTorch using the PyTorch Geometric Library [57] and the Deep Graph Library [86].

For the classifiers on data metrics (clustering coefficient, average degree, signal mean), we find the metrics for each sample/graph and use it as a feature for classification. Signal means is using the average across signal as the feature for each graph. # nodes and # of edges are using the number of nodes and number of edges, respectively. Clustering coefficient and average degree are taken for each graph. Combined is combining all of the metrics into a feature vector. We also use 10-fold cross-validation here with linear support vector machine, logistic regression (for single feature), and SGD Classifier (for twitch-egos there are more than 100,000 samples), all from scikit-learn library [87].

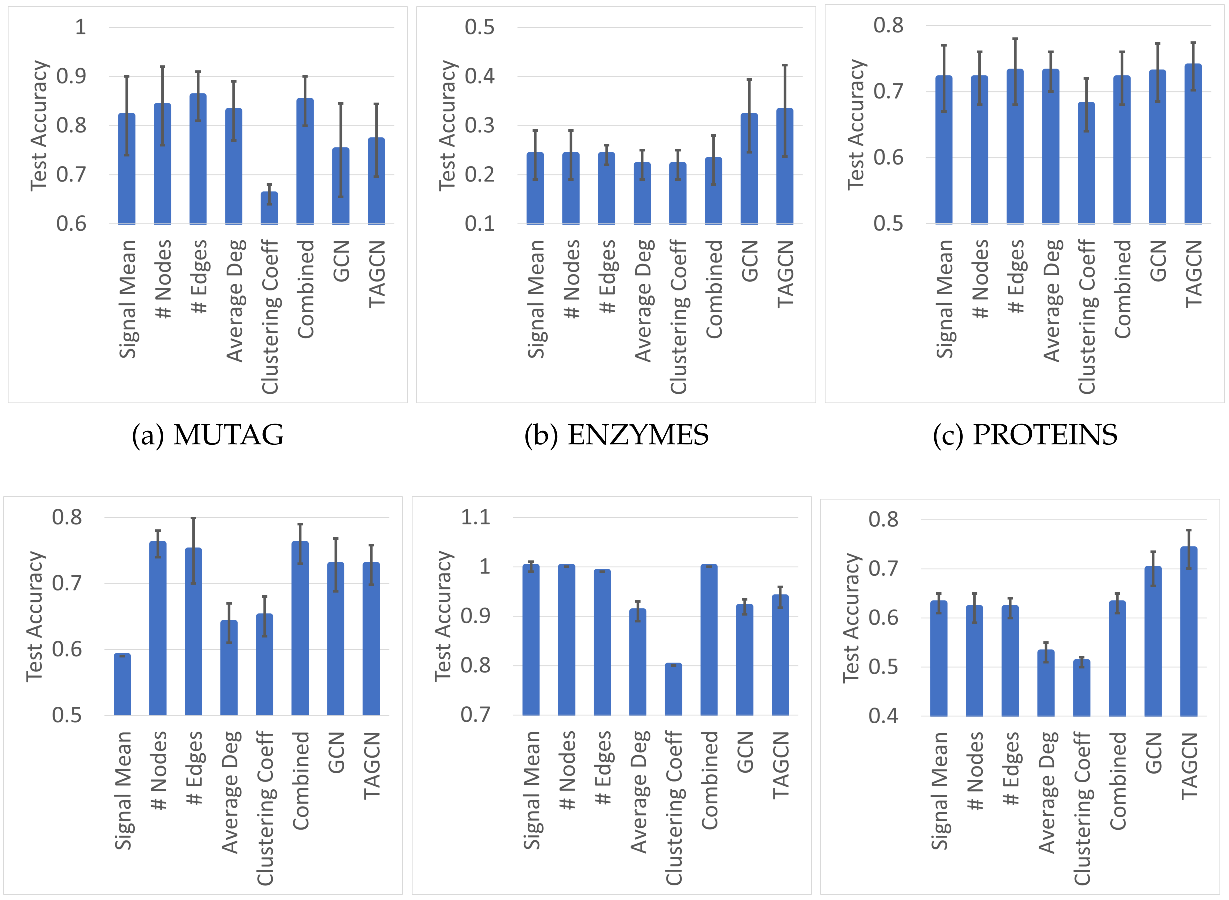
5.3.3 Results

Here, we report the results on 11 real graph classification datasets, quantified by test accuracy. We start with general GNN vs metrics results with a few simple GNN modifications inspired by the metrics results. We then focus more in-depth on the GNN architecture, i.e., number of layers, and order of polynomials.

5.3.3.1 Graph Neural Networks vs Metrics Results

Fig. 5.4 and 5.5 show the results of classifiers of different metrics (on graphs and/or signals) vs TAGCN for biological and social networks datasets, respectively. For some biological and social network datasets, classifiers based on simple network metrics/signal statistics lead to better or similar performance as TAGCN. For example, for MUTAG dataset, using the signal mean or number of edges in each graph as a feature with linear discriminant analysis outperforms the traditional neural network approach. For DD and AIDS datasets, either the number of nodes or the number of edges achieves better performance than TAGCN. For MUTAG, PROTEINS, and AIDS datasets, signal mean achieves good results. For IMDB-Binary and COLLAB, average degree performs well although not as good as GNN approaches. For ENZYMES, NCI1, Github stargazers and Twitch Egos datasets, GNN approaches have much better performance.

For signal, we have also looked into signal variance and higher order moments. For graphs, we have also looked into centrality, connectivity, and other metrics. We did not find better results than the metrics shown here.



(a) MUTAG (b) ENZYMES (c) PROTEINS

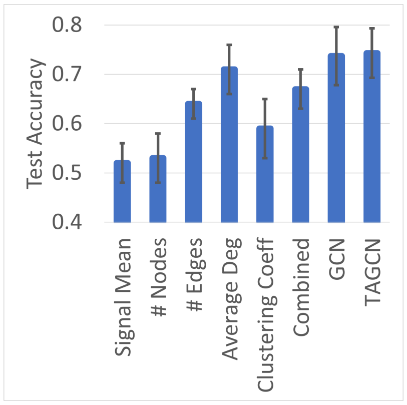
(d) DD (e) AIDS (f) NCI1

Figure 5.4: Data metrics vs GNN (TAGCN) for biological networks. Blue bars represent average test accuracy of each metric/architecture. Vertical black lines on the bars represent standard deviation.

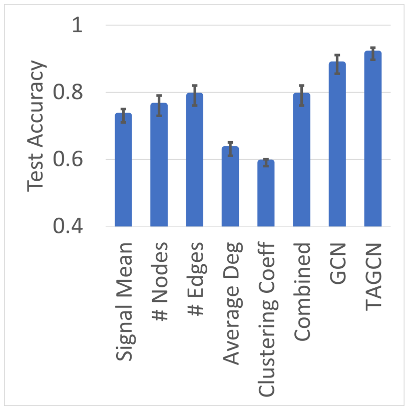
5.3.3.2 GNN Variant: Sumpooling

Fig. 5.4 and 5.5 show that for some datasets, using simple classifiers on the signal mean and number of nodes can perform well. For MUTAG and AIDS datasets, the simple classifiers outperform even default TAGCN. To better capture this using TAGCN, we modify the aggregation layer (Section 3.4) to use a sum pool instead of the default mean pool. Sumpool adds up the signal and hence would yield higher outputs for graphs with more nodes and higher signals.

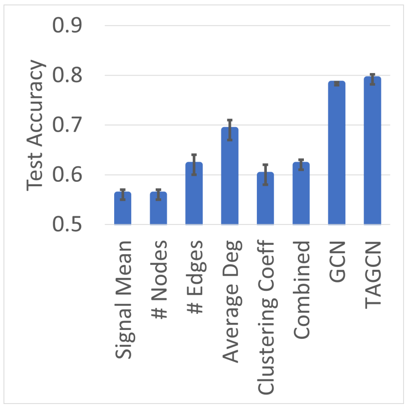
As shown in Fig. 5.6, we see that in general if the simple classifers on signal mean and/or number of nodes perform well, sumpool TAGCN performs better than meanpool TAGCN. Some notable examples are MUTAG, DD, and AIDS datasets, which have significant improvements using sumpooling over meanpooling. For AIDS dataset, the



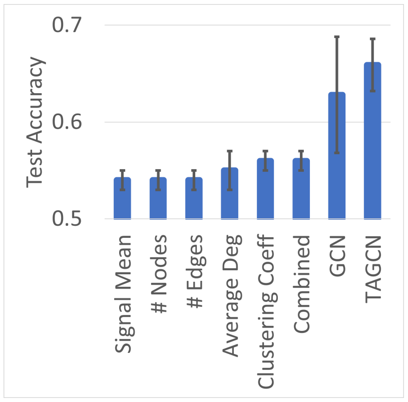
(a) IMDB-Binary



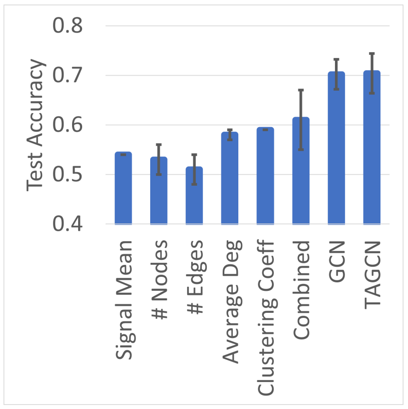
(b) Reddit-Binary



(c) COLLAB



(d) Github Stargazers



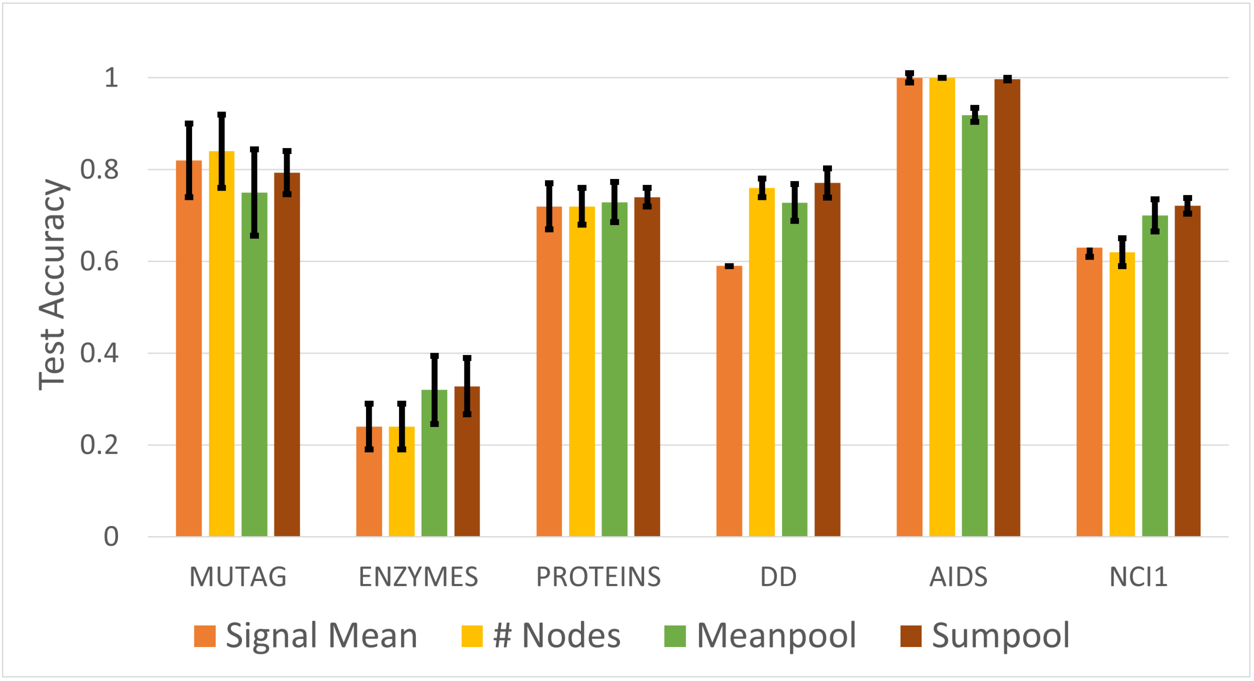
(e) Twitch Egos

Figure 5.5: Data metrics vs GNN (TAGCN) for social networks. Blue bars represent average test accuracy of each metric/architecture. Vertical black lines on the bars represent standard deviation.

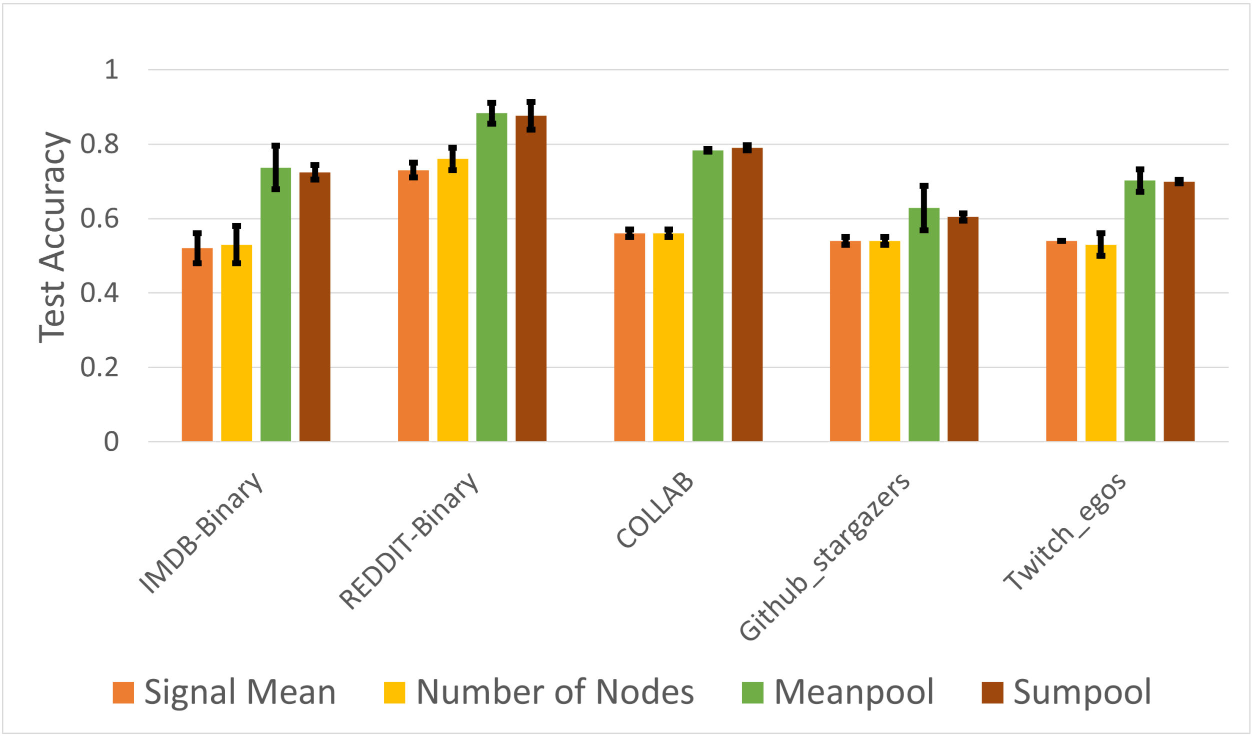
resulting improvement is close to 100% test accuracy as using either the signal means or the number of nodes to classify leads to 100% in accuracy. Surprisingly, default implementation of TAGCN (using meanpool) perform poorer than these simple metrics.

On the other hand, when using the simple classifiers on signal mean and/or number of nodes perform poorly, sumpool either does not lead to significant gain in test accuracy or perform a bit worse. This is seen in most of the social networks, where using signal mean and number of nodes have worse results than default TAGCN. For both IMDB-Binary and Reddit-Binary, there is a slight deterioration in performance.

In summary, sumpool is one simple variant that should be considered for GNN implementations before considering more complex models (that can capture the same information), especially if classifiers on the signals and/or number of nodes perform well.



(a) Biological networks



(b) Social networks

Figure 5.6: TAGCN (meanpool) vs TAGCN variant (sumpool) and data metrics for 5 real datasets. Colored bars represent average test accuracy of each metric/architecture. Vertical black lines on the bars represent standard deviation.

5.3.3.3 GNN Variant: Degree-aware GNN

Fig. 5.4 and 5.5 show that for some datasets, using simple classifiers on the number of edges and average degree can perform well, sometimes close to or better than GCN/TAGCN. To better capture this using GCN, we introduce 2 new variants called degree-aware GCN (DA-GCN) and degree-aware TAGCN (DA-TAGCN), modified from GCN (3.1), and TAGCN (3.2), respectively. Essentially, we add a learnable weight for each message-passing step that corresponds to the outdegree of each node. For GCN, the convolution becomes:

$$\mathbf{X}^{(\ell+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{V}^{(\ell)}\mathbf{X}^{(\ell)}\mathbf{W}^{(\ell)}\right). \tag{5.8}$$

where $\mathbf{V}^{(\ell)} = \mathrm{diag}(v_1^{(\ell)}, \cdots, v_N^{(\ell)})$ and $v_i^{(\ell)}$ is the learnable weight corresponding to the degree of node i at layer ℓ (shared weight for all nodes of the same degree in the same layer). Similarly, for TAGCN, we have shared weights for each degree of the polynomial filters:

$$X^{(\ell+1)} = \sigma\left(\sum_{k=0}^{K}\mathbf{A}^k\mathbf{V}_k^{(\ell)}X^{(\ell)}\mathbf{W}_k^{(\ell)}\right) \tag{5.9}$$

where $\mathbf{V}_k^{(\ell)} = \mathrm{diag}(v_{k,1}^{(\ell)}, \cdots, v_{k,N}^{(\ell)})$ and $v_{k,i}^{(\ell)}$ is the learnable weight corresponding to the degree of node i at layer ℓ and degree k (shared weight for all nodes of the same degree in the same layer and degree of the polynomial filters). For both GCN and TAGCN, these weights are initialized as 1's (so they have little effect on the loss and accuracy for the initial epochs of the training).

In Fig. 5.7, we show that using the DA-GCN and DA-TAGCN outperforms GCN and TAGCN when the number of edges and/or average degree is important for classification e.g., for MUTAG and PROTEINS datasets. On the other hand, when average degree is not as important, e.g., for DD, AIDS, and Reddit-Binary datasets, the performance using DA-GCN and DA-TAGCN drops. This is true especially for Reddit-Binary, where the performance drops significantly by 8% and 17% using DA-GCN and DA-TAGCN over GCN and TAGCN, respectively.

There is an anomaly in the results, for IMDB-Binary, simple classifiers using average degree performs well, but not DA-GCN and DA-TAGCN compared to GCN and TAGCN. A closer analysis tells us that this is because while IMDB-Binary dataset has similar average degrees between classes, but it has different degree distribution between classes.

In summary, DA-GCN and DA-TAGCN are two simple variants that can be considered for GNN implementations before considering more complex models (that can capture the same information), especially if classifiers on the average degree performs well. The amount of extra computation is small as the number of weights required is the highest degree in the dataset, or that number times the degree of polynomial filters for GCN and TAGCN, respectively.

5.3.3.4 GNN Variant: Graph Attention Networks

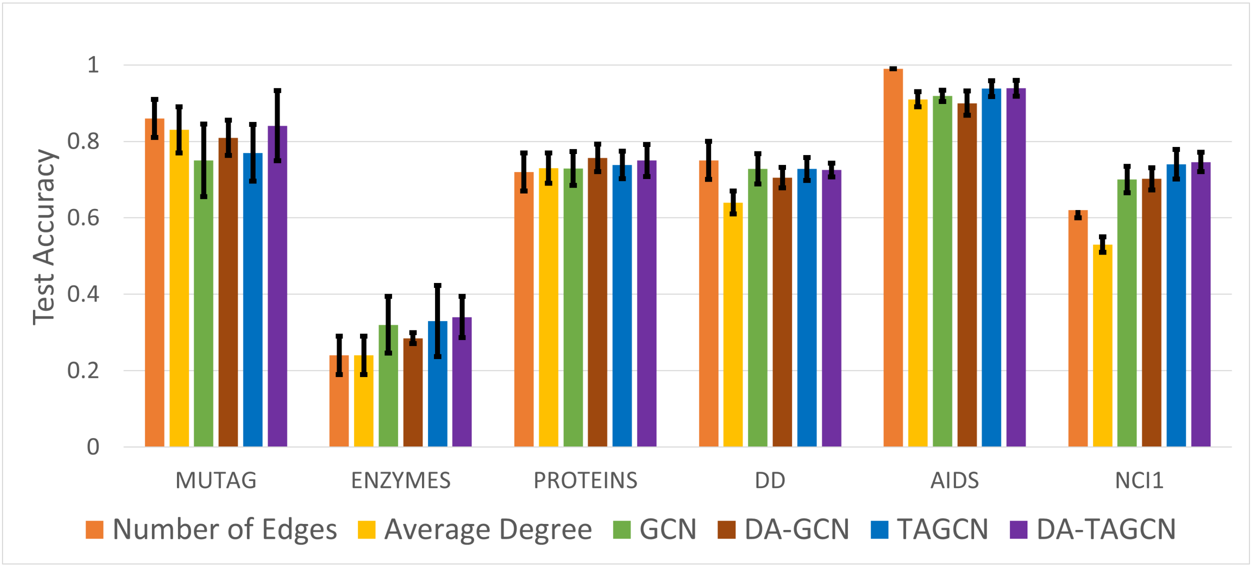
We also consider graph attention network (GAT) [50], one of the popular GNN variants that use self-attention to learn edge weights (see Section 4.3.3.3 for a short description). As shown in Fig. 5.8, GAT performs similar to TAGCN for all datasets (but better than GCN on some), as some might not expect with the additional complexity involving attention.

5.3.3.5 Impact of Graph Neural Network Architecture

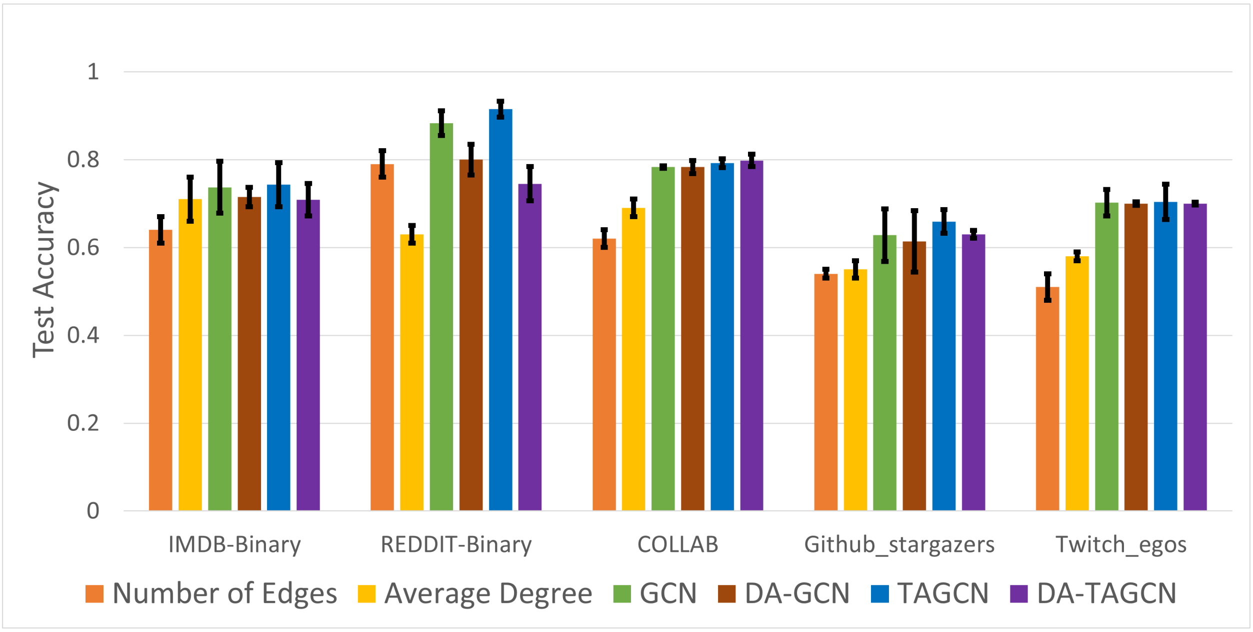
We look into two aspects of graph neural network architectures: number of layers and degree of the polynomial filters.

GCN: Number of Layers In general, GCN achieves optimal accuracy in less than 8 layers. Based on the data statistics in Tab. 5.2 and 5.3, it seems that datasets with high average degree and low average diameter require less number of layers to achieve optimal performance. Fig. 5.9 shows graph classification performance results versus number of graph convolutional layers for GCN for 11 selected real world datasets. In general, social datasets have lower average diameter than biological datasets. For social datasets, GCN performs optimally or close to optimal (within 1% of optimal) at 1-3 number of layers for these datasets. For the biological datasets, GCN perform optimally between 2-7 number of layers. The most noticeable social dataset is COLLAB, which has a high average degree, and GCN performs optimally at 1 layer. The most noticeable biological dataset is MUTAG, which has low average degree, and GCN performs optimally at 7 layers (it has similar accuracy at 3 layers, but the variance at 7 layers is lower).

We also see the effect of over-smoothing for GCN (see Section 4.2.4). For some biological datasets (Enzymes DD, NCI1), we see a significant drop in accuracy for layers 7-10. These datasets tend to have higher average diameter. Out of the the 5 social datasets, Reddit-Binary is the only one that experiences a significant drop in accuracy at the 10th



(a) Biological networks

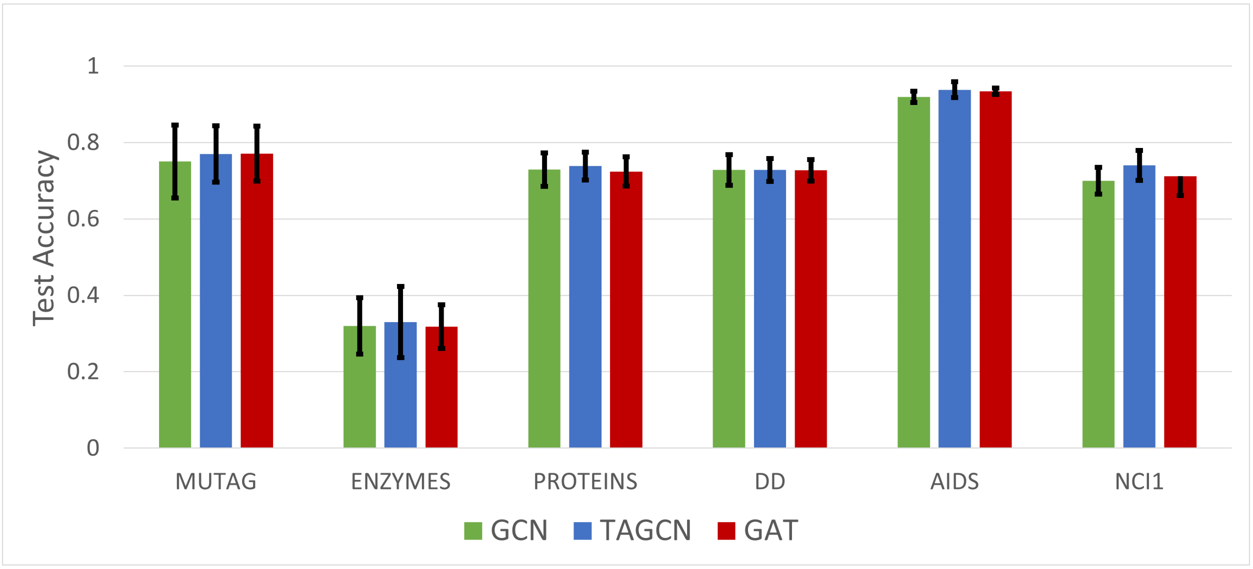


(b) Social networks

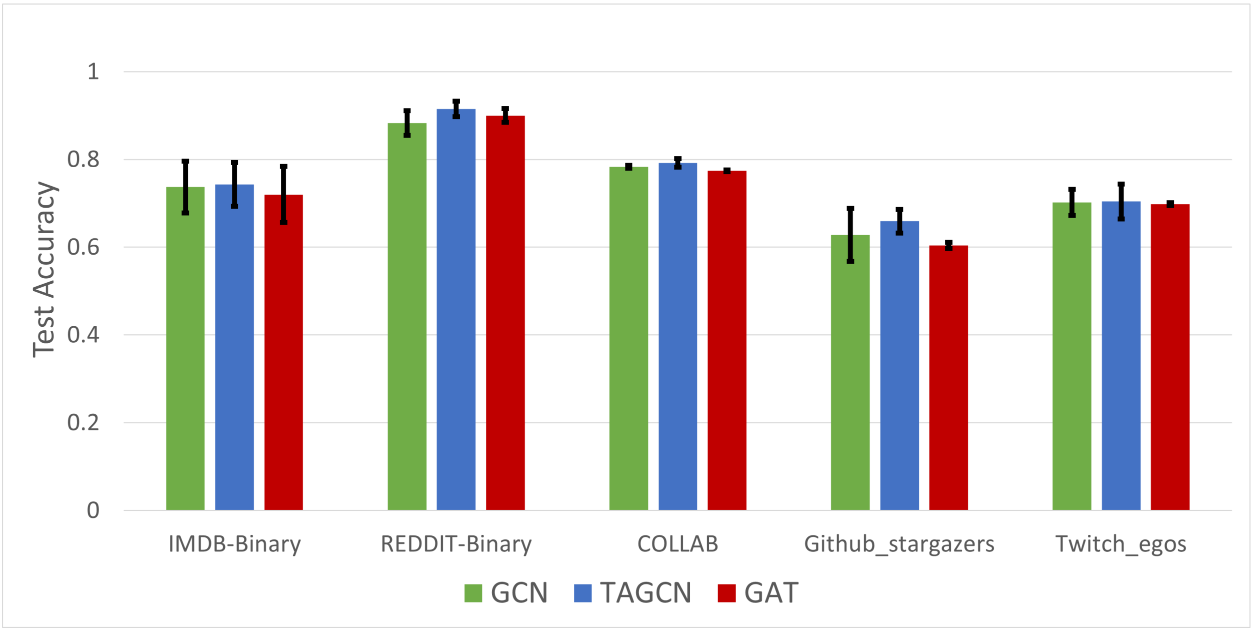
Figure 5.7: GCN and TAGCN vs GCN TAGCN variants (DA-GCN and DA-TAGCN) and data metrics for 5 real datasets. Colored bars represent average test accuracy of each metric/architecture. Vertical black lines on the bars represent standard deviation.

layer. It also incidentally has the highest average diameter and lowest average degree among the social datasets.

There are some exceptions to this. For example, Proteins dataset, which has higher average diameter than ENZYMES dataset, does not see a significant drop in accuracy even at the 10th layer for GCN. This might be due in part to the graph signal. We will



(a) Biological networks

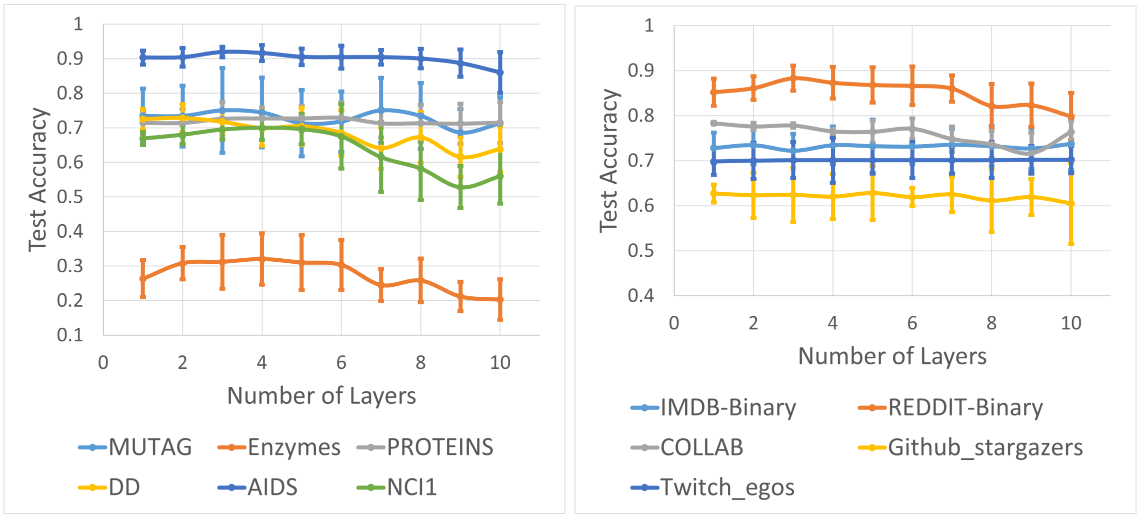


(b) Social networks

Figure 5.8: GCN and TAGCN vs GNN variant (GAT) for 5 real datasets. Colored bars represent average test accuracy of each architecture. Vertical black lines on the bars represent standard deviation.

investigate these trends further using synthetic datasets.

For 3 social datasets (IMDB-Binary, Twitch_egos, and Github_stargazers), GCN performs close to the optimal accuracy even at 10 layers. This is because the over-smoothed signals of the distinguishable graphs are far apart, so more convolutional layers do not converge them.

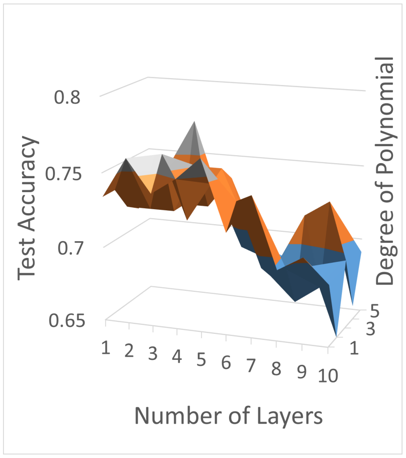


(a) Biological networks (b) Social networks

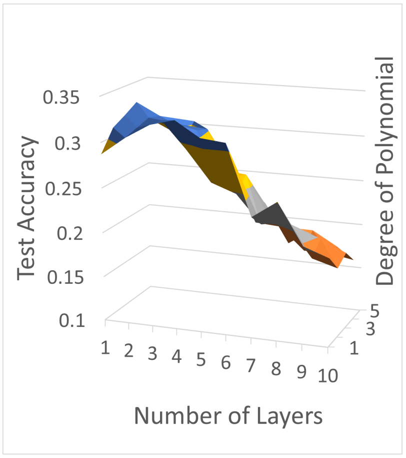
Figure 5.9: GCN: Test accuracy vs number of layers for 12 real datasets. Colored horizontal and vertical lines represent average test accuracy and standard deviation of each dataset, respectively.

TAGCN: Number of Layers and Degree of the Polynomial Filters Fig. 5.10 and 5.11 show graph classification performance results versus number of graph convolutional layers for TAGCN for biological and social datasets, respectively. TAGCN uses polynomial filters up to degree 3. TAGCN is more general than GCN and reduces to GCN by taking the degree of the polynomial filter $P(\mathbf{A})$ to be $K = 1$ and removing the residual connection. In general, we see similar trend as GCN in accuracy with respect to the number of layers. For all datasets, IMDB-Binary, Reddit-Binary, Twitch_egos, and Github_stargazers, the accuracy drops at higher number of layers. Reddit-Binary's accuracy drops slower vs number of layers with TAGCN compared to GCN.

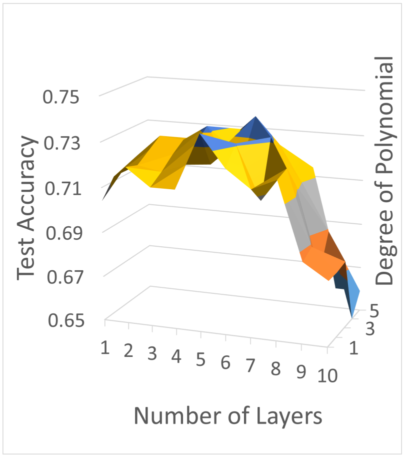
There is no clear result as to the optimal degree of the polynomial filters for different datasets and number of layers. Like GCN, the variations seem to be small relative to number of layers for some datasets (Github Stargazers and Twitch Egos). For some datasets (e.g., NCI1 and IMDB-Binary), TAGCN performs better with lower degree of the polynomial filters at higher layers. And for some datasets with low average degree (e.g., AIDS, Reddit-Binary), TAGCN performs better with higher degrees of the polynomial filters at lower layers.



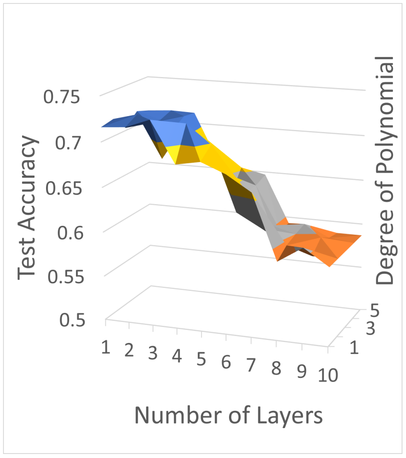
(a) MUTAG



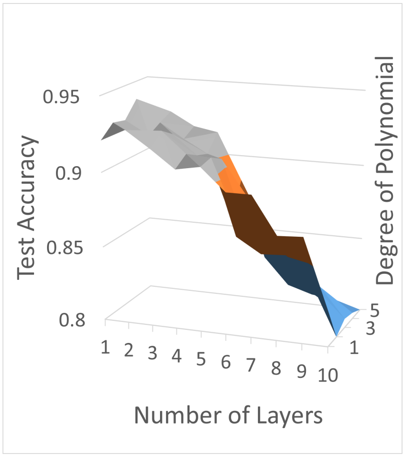
(b) ENZYMES



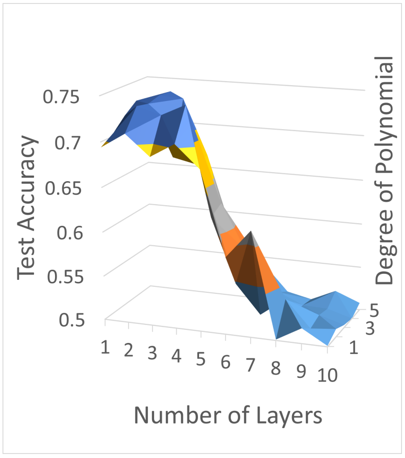
(c) Proteins



(d) DD



(e) AIDS

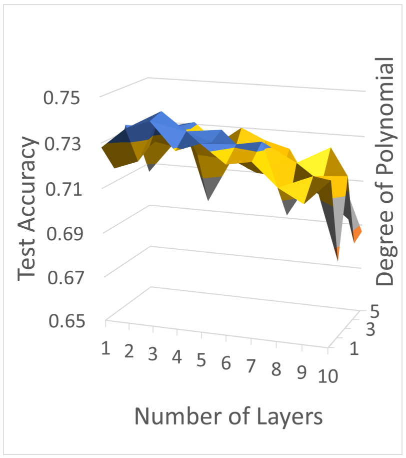


(f) NCI1

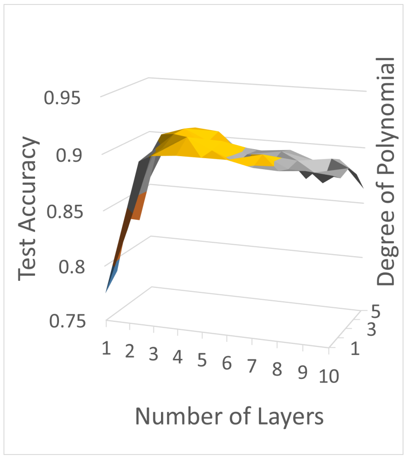
Figure 5.10: TAGCN: Test accuracy vs number of layers and degree of the polynomial filters for 6 real biological datasets.

5.4 Synthetic Data

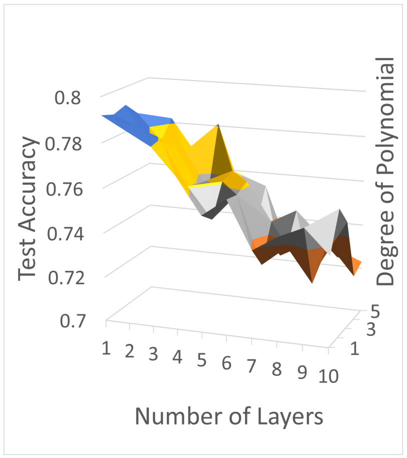
Compared to real datasets, synthetic datasets allow for a more comprehensive analysis of the impact of graph structure on the performance of the GNN architecture topology. Just like for graph classification datasets, this is because 1) with synthetic datasets, we can systematize the graphs and study the impact of each characteristic of the graph at a time, and 2) we can generate many dataset samples to draw more reliable conclusions.



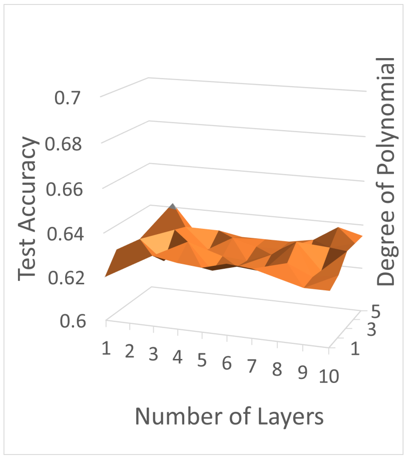
(a) IMDB-Binary



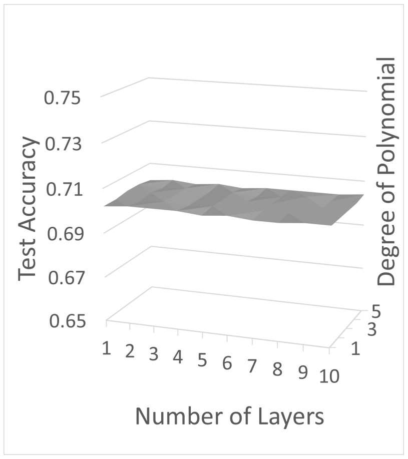
(b) Reddit-Binary



(c) COLLAB



(d) Github Stargazers



(e) Twitch Egos

Figure 5.11: TAGCN: Test accuracy vs number of layers and degree of the polynomial filters for 5 real social datasets.

5.4.1 Data Description

We focus on 3 types of graphs that are commonly studied: Erdős-Rényi, smallworld, and preferential attachment. See section 4.4.1 for full descriptions.

5.4.2 Experimental Setup

We follow the same experimental setup as that for real datasets (see Section 5.3.2). For default graph generation setup, we find parameters that lead to distinguishable results. Unless mentioned otherwise, we use:

- 2 classes, 0 and 1

- 1000 graphs, each with 100-500 nodes (randomly chosen)
- Graph signal x is a 1-Dimensional feature vector with $N(\mu_{Y_i}, 1)$ where Y_i is the class of graph i. The class means μ_0 and μ_1 are taken to be $\mu_{signal} = 0$.
- For Erdős-Rényi, we generate graphs with following edge creation probability $p_i = r_{Y_i} \frac{\log N_i}{N_i}$ where N_i is the number of nodes for sample graph i, $r_0 = 3$ for class 0 and $r_1 = 3.8$ for class 1 .
- For Watts–Strogatz, we generate graphs with the same mean degree $\kappa_1 = \kappa_2 = 20$ and rewiring probability p_{Y_i} of 0.3 for class 0 and 0.2 for class 1.
- For Barabási–Albert, we generate graphs with edges-to-attach $m_0 = 10$ for graphs of class 0 and $m_1 = 12$ for graphs of class 1.

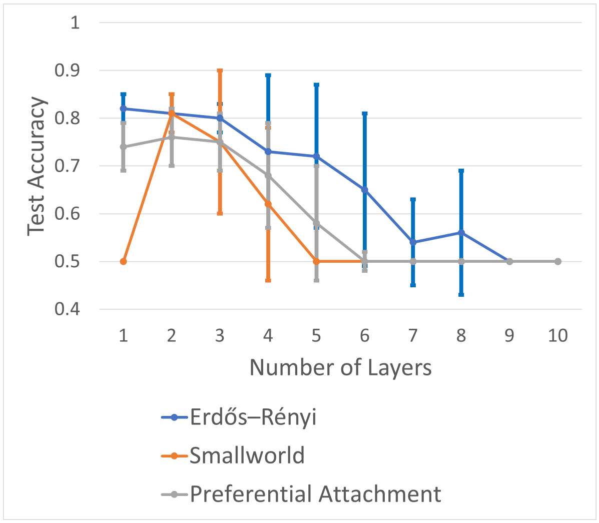
5.4.3 Results

We report the results of several characteristics on model performance for 3 types of synthetic graph classification networks, quantified by test accuracy. For the architecture, we focus on the number of layers and the degree of the polynomial filters.

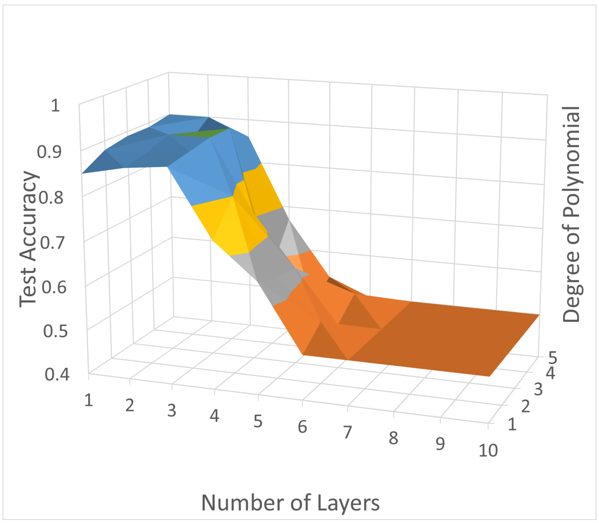
5.4.3.1 General Results

We first discuss general results comparing the 3 network models. Fig 5.12a shows the results of GCN for different number of layers. Both Erdős-Rényi and preferential attachment networks perform well with 1-3 layers while smallworld's accuracy becomes relevant only at 2 layers (it performs randomly with a single layer). All networks eventually succumb to over-smoothing, with smallworld having the sharpest drop in accuracy starting at layer 3 and dropping to random at layer 5 followed by preferential attachment at layer 4 before dropping to random at layer 6. Erdős-Rényi's drop is more gradual from layer 4 to layer 9.

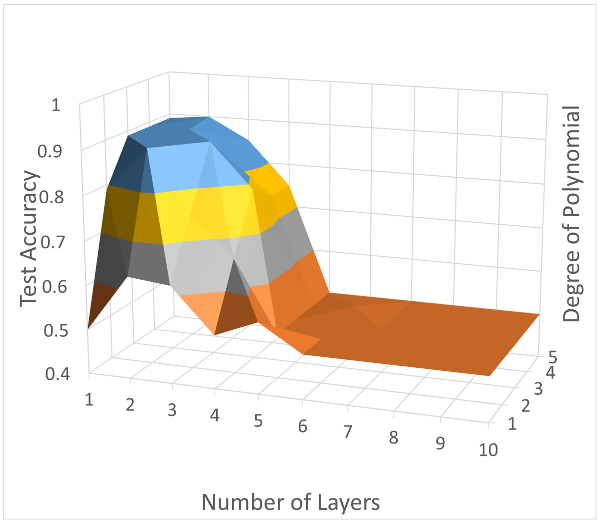
Other subfigures of Fig. 5.12 show the results of TAGCN for different number of layers and degree of the polynomial filters. The degree of the polynomial filters has different effect for different network models. For Erdős-Rényi (Fig. 5.12b), higher degrees are slightly preferable for layers 1-3, while lower degrees are preferable for layers 5-6. For smallworld (Fig. 5.12c), the filters play a major role in the first 4 layers (for layer 1, accuracies are 0.5, 0.79, 0.89, 0.89, 0.89 for degrees of the polynomial filters from 1 to 5). For preferential attachment (Fig. 5.12d), the plot are similar to Erdős-Rényi's but



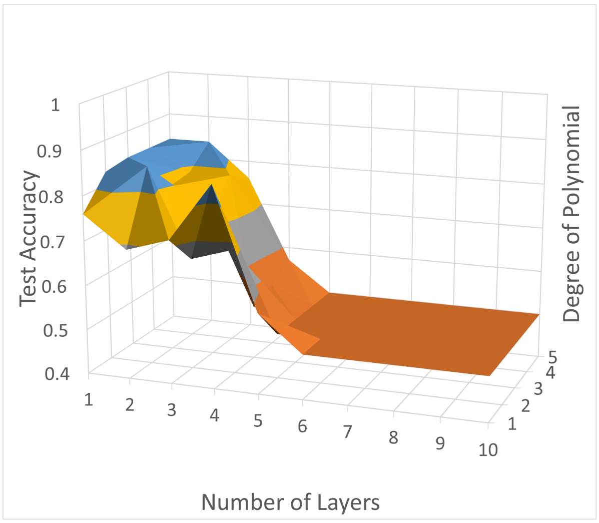
(a) GCN: All 3 models of synthetic graphs. Colored horizontal and vertical lines represent average test accuracy and standard deviation of each model, respectively.



(b) TAGCN: Erdős-Rényi



(c) TAGCN: Smallworld



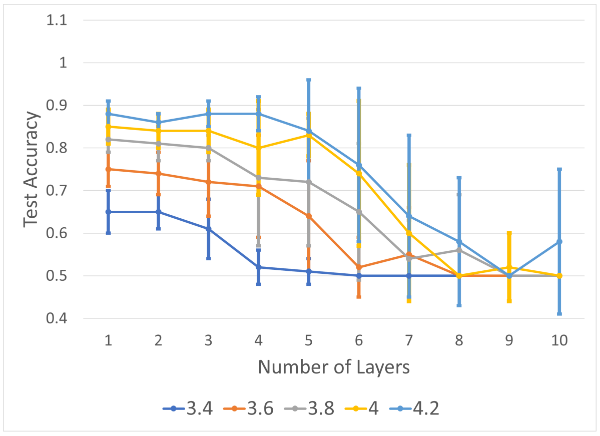
(d) TAGCN: Preferential Attachment

Figure 5.12: General GCN and TAGCN: Test accuracy vs number of layers and/or degree of the polynomial filters for 3 models of synthetic graphs.

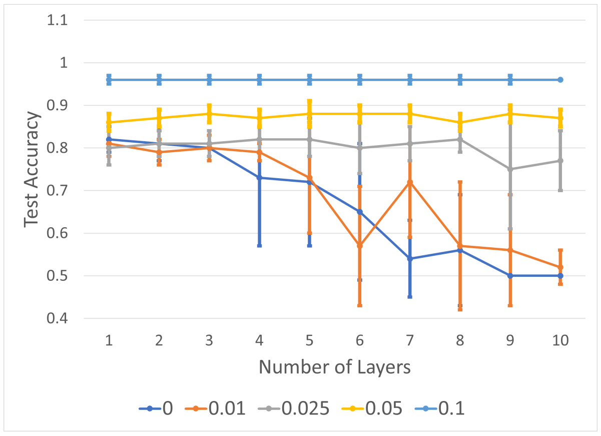
less smooth: higher degree of the polynomial filters is helpful again for layers 1-3, while lower degree is better for layers 4-5.

5.4.3.2 Erdős-Rényi Network Results

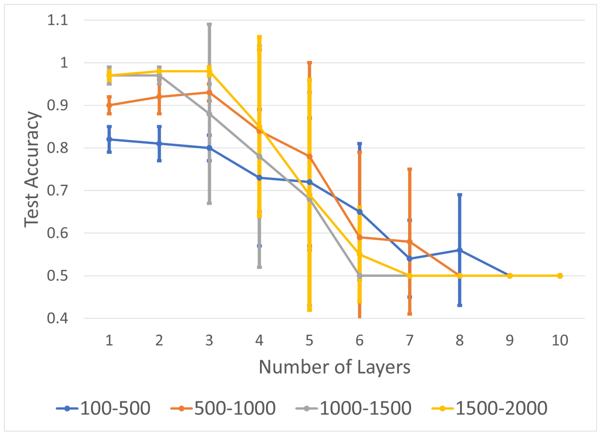
Fig. 5.13 and 5.14 show the results of GCN and TAGCN vs number of layers, number of layers + degree of the polynomial filters, respectively for different Erdős-Rényi graph



(a) r_1 (coefficient of class 1 edge creation probability)



(b) μ_{signal}

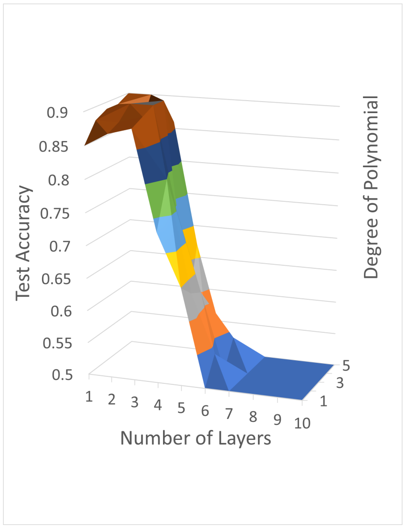


(c) Number of nodes

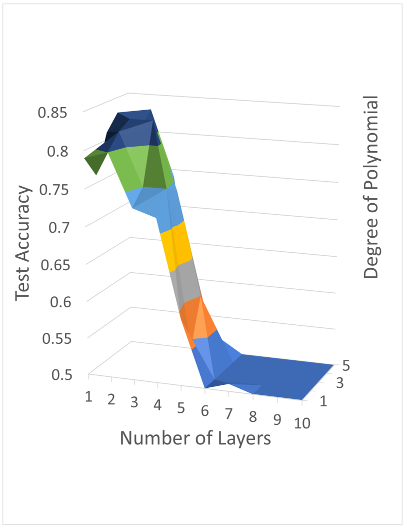
Figure 5.13: GCN: Test accuracy vs number of layers and different Erdős-Rényi model generation parameters. Subcaption indicates the model generation parameter that is changed for different colored lines. Colored horizontal and vertical lines represent average test accuracy and standard deviation, respectively.

characteristics. The graph characteristics are: edge creation probability p, graph signal, and number of nodes.

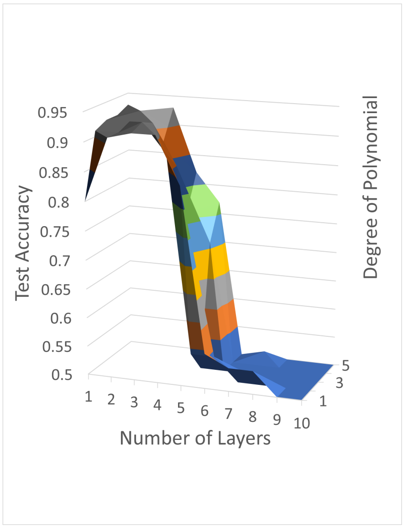
Impact of Edge Creation Probability Fig. 5.13a shows the results of GCN vs number of layers for different coefficient of class 1 edge creation probability r_1 (with r_0 fixed at 3). For each graph $p_i = r_{Y_i}\frac{\log N_i}{N_i}$ where N_i and Y_i are the number of nodes, and the class of graph sample i (see Section 5.4.2 for the experimental setup). In general, as this coefficient increases, the test accuracy improves. The number of layers where it plateaus near optimal accuracy increases as well (with r_1 of 4 and 4.2 having high accuracy for



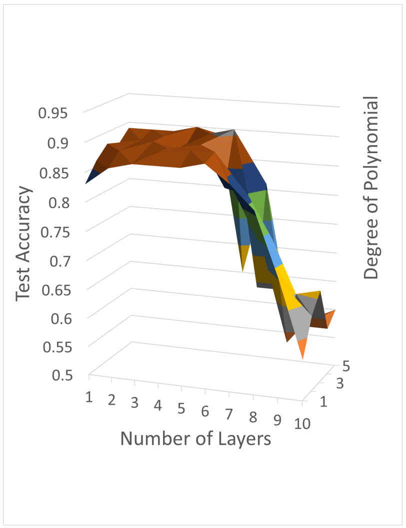
(a) Default (coefficients for edge creation probabilities: $r_0 = 3$, $r_1 = 3.8$, mean of signals, $\mu_{\text{signal}} = 0$, number of nodes=100-500)



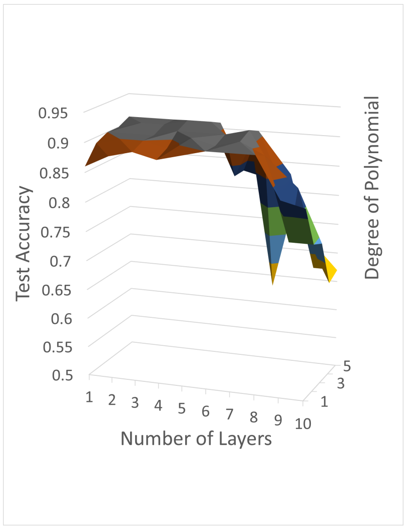
(b) $r_1 = 3.6$



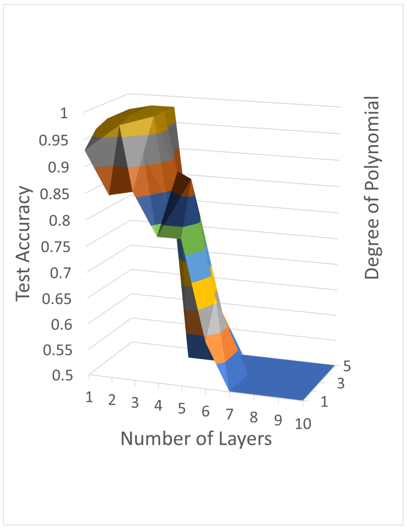
(c) $r_1 = 4.0$



(d) $\mu_{\text{signal}} = \pm 0.025$



(e) $\mu_{\text{signal}} = \pm 0.05$



(f) Number of nodes=500-1000

Figure 5.14: TAGCN: Test accuracy vs degree of the polynomial filters, number of layers, and different Erdős-Rényi model generation parameters. Subcaption indicates the model generation parameter that is changed from the default setup.

the first 5 layers and r_1 or 3.4 starting to drop in accuracy at the 3rd layer).

Fig. 5.14b, 5.14a, 5.14c show the results of TAGCN vs number of layers and degree of polynomial filters for 3.6, 3.8, and 4.0 coefficients of class 1 edge creation probability r_1, respectively. Like GCN, in general, as this coefficient (and hence edge creation probability) increases, the test accuracy improves across both layers and degree of polynomial filters. The effect of the number of layers is similar to GCN, while the degree of polynomial filters seem to have a higher optimal value for lower layers and lower for higher layer, e.g., for the default ($r_1 = 3.8$) case for layers 1-6, the optimal degrees of the polynomial filters are 5, 3, 3, 2, 1, 2.

Impact of Graph Signals Fig. 5.13b shows the results of GCN vs number of layers for different class means of signals. As the mean increases, the accuracy improves as expected. When it reaches a critical value between 0.01 and 0.025, the test accuracy smoothes out and the GCN is able to resist the over-smoothing effect for the first 10 layers. Further increases in the mean simply lead to higher accuracy.

Fig. 5.14a, 5.14e show the results of TAGCN vs number of layers and degree of polynomial filters for ± 0, ± 0.025, and ± 0.05 class means of signals, respectively. Like GCN, in general, as the difference in means of signals increases, the test accuracy increases across both layers and degree of polynomial filters. Similar to the default case, we still see higher degree of polynomials are better at lower layers and vice versa for higher layers. For example, for $\mu_{\text{signal}} = \pm 0.025$, the optimal degrees of the polynomial filters for layers 1-10 is: 5,5,3,5,4,4,1,1,1,3.

At higher signals, the plots flatten out and there is little difference in terms of both the number of layers up to 10 and degree of the polynomial filters up to 5. The shape looks similarly to the 3D surface plots of Github_stargazers and Twitch_egos in Fig. 5.11. We suspect that the signals for these two datasets are probably more distinct than other datasets between graphs. However, since they are not 100% accuracy, they are not split cleanly between classes, i.e., if there are 2 classes, then there is perhaps 2 binomdal distributions, with a 70% chance of class 0 and 30% of class 1 for the first bimodal and vice versa for the second bimodal.

Impact of Number of Nodes Fig. 5.13c shows the results of GCN vs number of layers for different number of nodes. As the number of nodes increases, the accuracy improves. The accuracy for all values of the number of nodes reaches maximum within the first 3 layers, then drop to random. There is a large variance when accuracy starts dropping after the initial layers.

Fig. 5.14a, 5.14f show the results of TAGCN vs number of layers and degree of polynomial filters for 100-500, 500-1000 number of nodes, respectively. Like GCN, in general, as the number of nodes increases, the accuracy improves across both layers and degree of polynomial filters.

5.4.3.3 Smallworld Network Results

Fig. 5.15 and 5.16 show the results of GCN and TAGCN vs number of layers, number of layers + degree of the polynomial filters, respectively for different smallworld graph characteristics. The graph characteristics are: rewiring probability p, mean degree k, graph signal, and number of nodes.

Impact of Edge Rewiring Probability p_1 Fig. 5.15a shows the results of GCN vs number of layers for different class 1 edge rewiring probability p_1. As the probability gets further away from class 0 ($p_0 = 0.3$), the accuracy increases. GCN performs better with decreased probability over increased probability, in terms of both accuracy and over-smoothing. The shapes of the curves are related to the increase in accuracy vs p_0, with increase in resistance to over-smoothing as the p_1 differs more from p_0.

Fig. 5.16a, 5.16b, 5.16c show the results of TAGCN vs number of layers and degree of polynomial filters for 0.2, 0.4, 0.6 class 1 edge rewiring probability p_1, respectively. Like GCN, in general, as the probability differs more from $p_0 = 0.3$, the test accuracy increases across both layers and degree of polynomial filters. In all cases, for the initial layers, TAGCN performs very poorly when the degree of polynomial filters is 1 and performs better when the degree is higher.

Impact of Mean Degree κ Fig. 5.15b shows the results of GCN vs number of layers for different class 1 mean degree κ_1 while fixing the edge rewiring probability $p_1 = p_0 = 0.3$. Since GCN performs well even at 1 layer, this shows that it is the difference in edge rewiring probability that leads to the drop at first layer (and not difference in mean degree). This suggests that Reddit-Binary dataset, which also requires at least 2 layers to perform well, may have smallworld characteristics with and different edge rewiring probability for different classes.

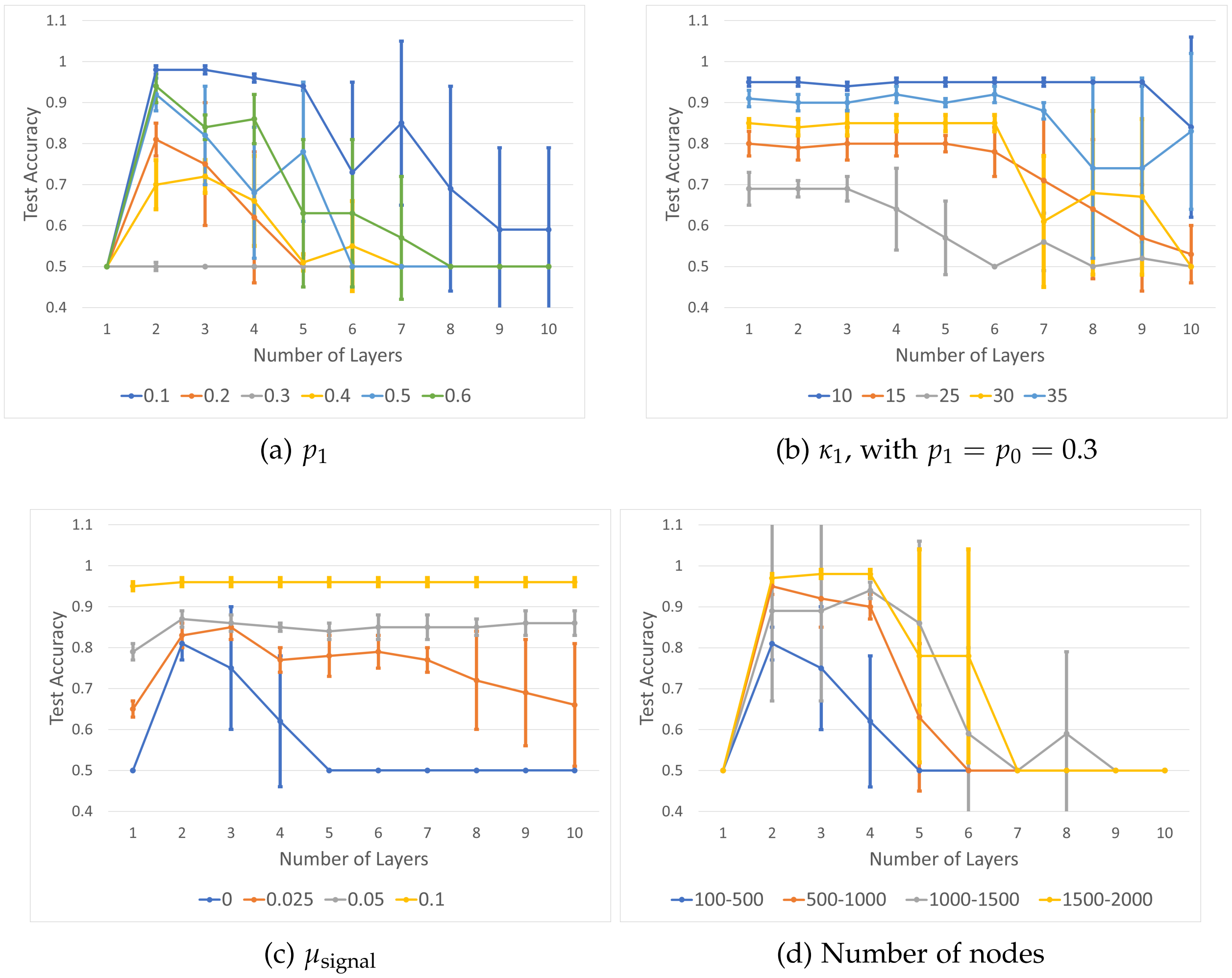
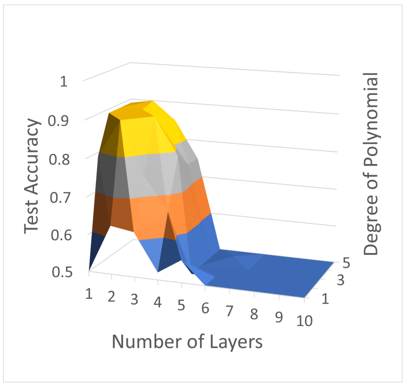


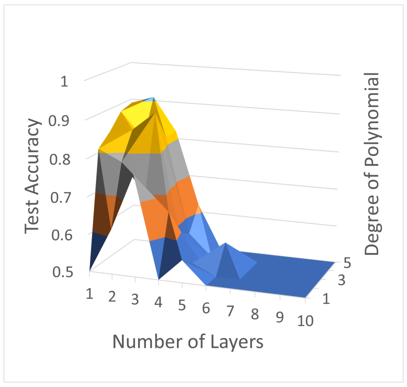
Figure 5.15: GCN: Test accuracy vs number of layers and different smallworld model generation parameters. Subcaption indicates the model generation parameter that is changed for different colored lines. Colored horizontal and vertical lines represent average test accuracy and standard deviation, respectively.

As the mean gets further away from class 0 ($\kappa_0 = 20$), the accuracy increases. GCN performs better with decreased mean over increased mean, in terms of both accuracy and over-smoothing, suggesting that less small-worldness property is easier to differentiate. Surprisingly, regardless of the mean (as long $|\kappa_1 - \kappa_0| > 5$), the initial 3 layers are reasonably stable in terms of accuracy (initial 5 layers for all except $\kappa_1 = 25$).

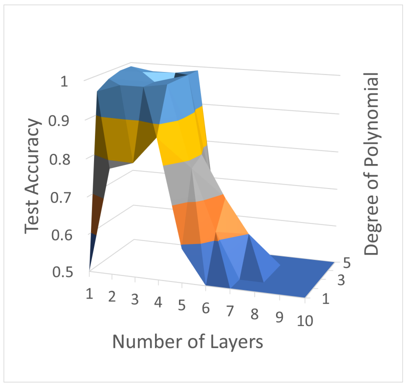
Fig. 5.16b and 5.16c show the results of TAGCN vs number of layers and degree of polynomial filters for 15 and 25 class 1 mean degree κ_1, respectively (while fixing the edge rewiring probability $p_1 = p_0 = 0.3$). TAGCN on $\kappa_1 = 15$ is better than $\kappa_1 = 25$ (when $\kappa_0 = 20$), suggesting also that less small-worldness property is more distinguishable by TAGCN. For lower $\kappa_1 = 15$, TAGCN performs well with degree of polynomial



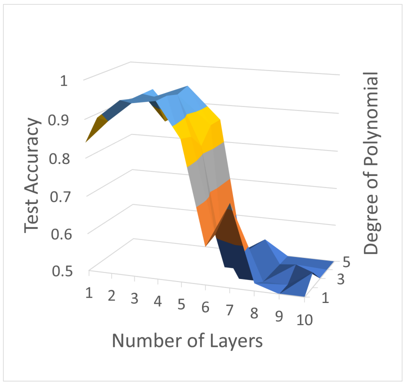
(a) Default (edge rewiring probabilities $p_0 = 0.3$, $p_1 = 0.2$, mean degrees $\kappa_0 = \kappa_1 = 20$, $\mu_{\text{signal}} = 0$, number of nodes = 100-500)



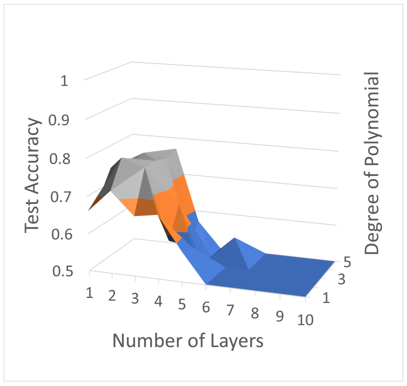
(b) $p_1 = 0.4$



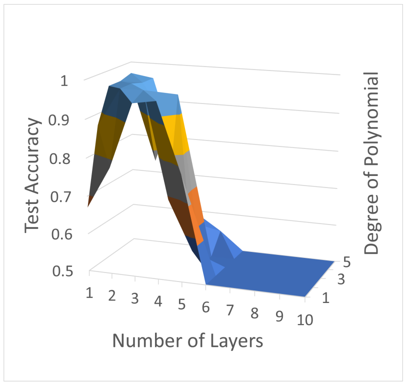
(c) $p_1 = 0.6$



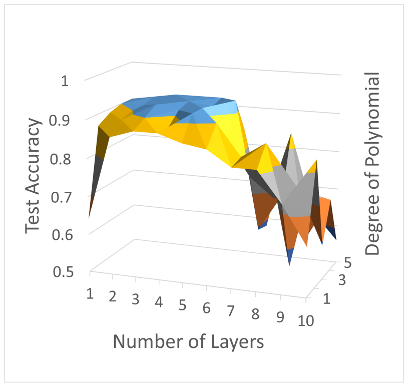
(d) $\kappa_1 = 15, p_1 = 0.3$



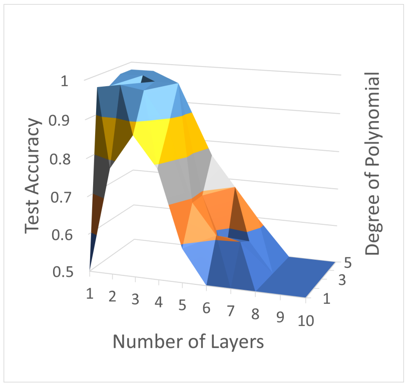
(e) $\kappa_1 = 25, p_1 = 0.3$



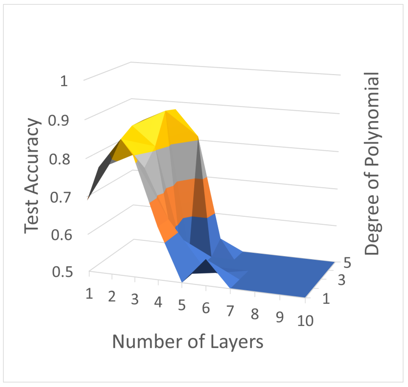
(f) $\kappa_1 = 25$, $p_1 = 0.2$



(g) $\mu_{\text{signal}} = 0.025$



(h) number of nodes=500-1000



(i) number of nodes=500-1000, , $p_1 = 0.3$, $\kappa_1 = 25$

Figure 5.16: TAGCN: Test accuracy vs number of layers, degree of the polynomail filters, and different smallworld model generation parameters. Subcaption indicates the model generation parameter that is changed from the default setup.

filters of 1 for the initial layers (where as for the $\kappa_1 = 25$ and default case, 1 is not good).

Impact of Graph Signal Fig. 5.15c shows the results of GCN vs number of layers for different class means of signals. As the mean increases, the accuracy improves as expected. When it reaches a critical value between 0 and 0.025, the test accuracy smoothes out and the GCN is able to resist the over-smoothing effect for the first 10 layers. Further increases in the mean simply lead to higher accuracy.

Fig. 5.16a, 5.16g show the results of TAGCN vs number of layers and degree of polynomial filters for ± 0 and ± 0.025 class means of signals, respectively. Like GCN, in general, as the difference in means of signals increases, the test accuracy increases across both layers and degree of polynomial filters. Just like the default case, we still see higher degree of polynomials are better at lower layers and vice versa for higher layers.

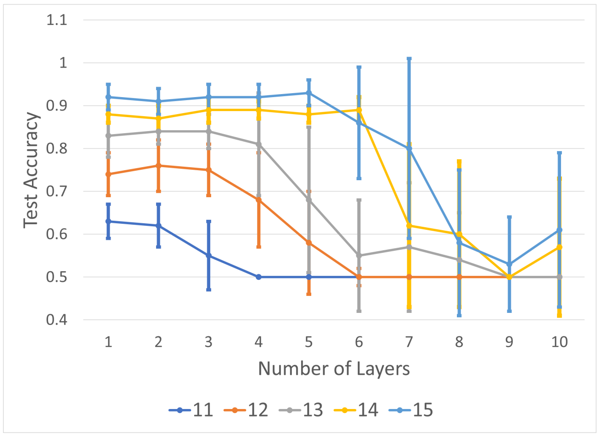
Impact of Number of Nodes Fig. 5.15d shows the results of GCN vs number of layers for different number of nodes. As the number of nodes increases, the accuracy improves. For all number of nodes, GCN does not perform well with just 1 layer (GCN needs 2 layers to differentiate rewiring probability).

Fig. 5.16h shows the results of TAGCN vs number of layers and degree of polynomial filters for 500-1000 number of nodes. Like GCN, as the number of nodes increases, the accuracy improves across both layers and degree of polynomial filters. The surface plot is similar to the default 100-500 number of nodes case with lower performance for 1 degree of the polynomial filters.

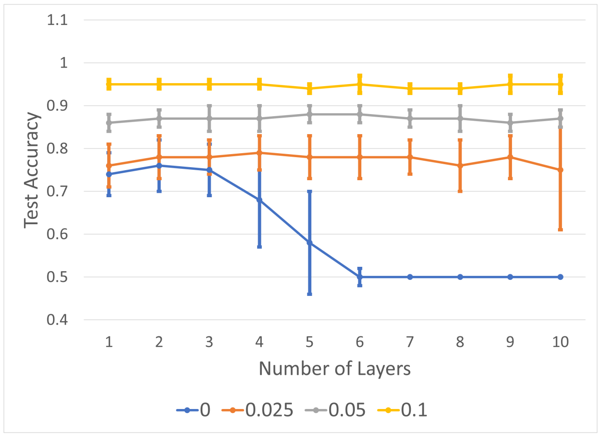
Fig. 5.16i shows the results of TAGCN vs number of layers and degree of polynomial filters for 500-1000 number of nodes with same rewiring probability $p_1 = p_0 = 0.3$ but different mean degree $\kappa_1 = 25$. Unlike the previous case, we see that TAGCN performs better for 1 degree of the polynomial filters. This suggests that to detect difference in edge rewiring probability using TAGCN, it is best to have at least 2 layers.

5.4.3.4 Preferential Attachment Network Results

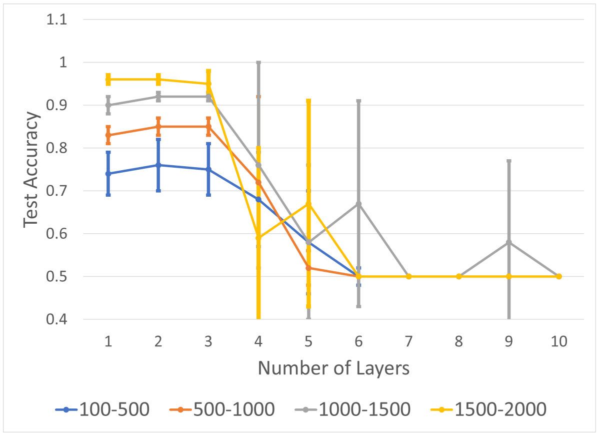
Fig. 5.17 and 5.18 show the results of GCN and TAGCN vs number of layers, number of layers + degree of the polynomial filters, respectively for different preferential attachment graph characteristics. The graph characteristics are: edges-to-attach m, graph signal, and number of nodes.



(a) Edges-to-attach m_1



(b) μ_{signal}

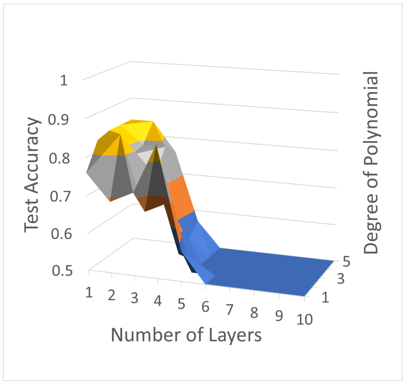


(c) Number of nodes

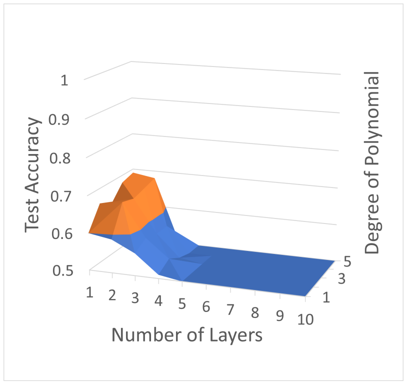
Figure 5.17: GCN: Test accuracy vs number of layers and different preferential attachment model generation parameters. Subcaption indicates the model generation parameter that is changed for different colored lines. Colored horizontal and vertical lines represent average test accuracy and standard deviation, respectively.

Impact of Edges-to-attach m Fig. 5.17a shows the results of GCN vs number of layers for different class 1 edges-to-attach m_1. As m_1 increases, accuracy improves. The plot seems similar to Erdős-Rényi class edge creation probability r_1 (see Fig. 5.13a). Being preferentially attached has little effect.

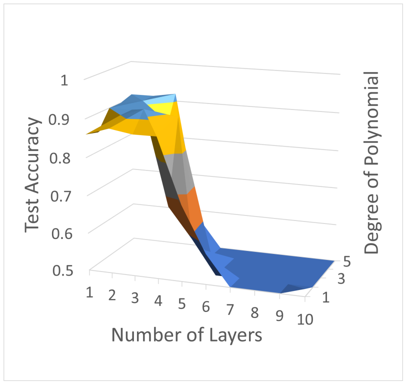
Fig. 5.18b, 5.18a, and 5.18c show the results of TAGCN vs number of layers and degree of polynomial filters for 11, 12, 13 edges-to-attach m_1, respectively. Just like GCN, TAGCN performs better as m_1 increases. At $m_1 = 13$, the surface plot looks similar to TAGCN for Erdős-Rényi (see Fig. 5.14c).



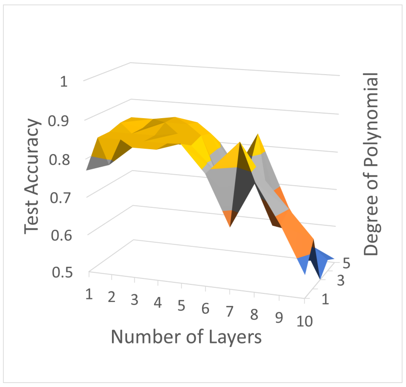
(a) Default (edges-to-attach $m_0 = 10$ and $m_1 = 12$, $\mu_{\text{signal}} = 0$, number of nodes = 100-500)



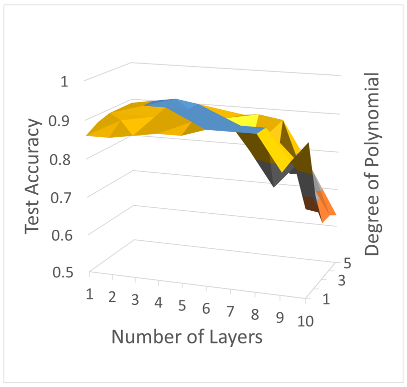
(b) $m_1 = 11$



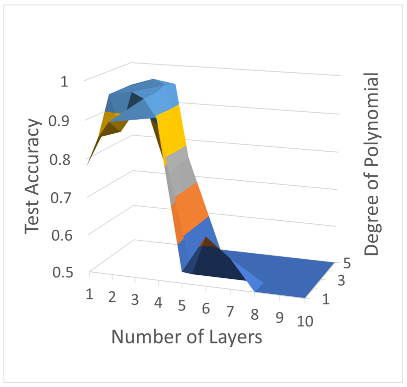
(c) $m_1 = 13$



(d) $\mu_{\text{signal}} = \pm 0.025$



(e) $\mu_{\text{signal}} = \pm 0.05$



(f) Number of nodes = 500

Figure 5.18: TAGCN: Test accuracy vs number of layers, degree of the polynomail filters, and different preferential attachment model generation parameters. Subcaption indicates the model generation parameter that is changed from the default setup.

Impact of Graph Signals Fig. 5.17b shows the results of GCN vs number of layers different class means of signals. As the mean increases, accuracy improves. The plot is similar to signal plot of Erdős-Rényi (see Fig. 5.13b).

Fig. 5.18a, 5.18d, and 5.18e show the results of TAGCN vs number of layers and degree of polynomial filters for ± 0, ± 0.025, and ± 0.05 class means of signals, respectively. Like GCN, in general, as the difference in means of signals increases, the test accuracy increases across both layers and degree of polynomial filters. Again, these are similar to TAGCN plots for ± 0.025, and ± 0.05 for Erdős-Rényi (see Fig. 5.14d, and 5.14e).

Impact of Number of Nodes Fig. 5.17c shows the results of GCN vs number of layers for different number of nodes. As the number of nodes increases, the accuracy improves.

Just like Erdős-Rényi (see Fig. 5.13c), the accuracy for all number of nodes reaches maximum within the first 3 layers, then drop to random. There is significant of variance when accuracy starts dropping after the initial layers. Unlike Erdős-Rényi, the drop in accuracy seems to be sharper with high variance.

Fig. 5.18a, 5.18f show the results of TAGCN vs number of layers and degree of polynomial filters for 100-500, 500-1000 number of nodes, respectively. Like GCN, in general, as the number of nodes increases, the accuracy improves across both layers and degree of polynomial filters. The plots again look similar to TAGCN for number of nodes plots of Erdős-Rényi (see Fig. 5.14f), but smoother.

5.4.3.5 GNN Variants: GAT and DA-TAGCN

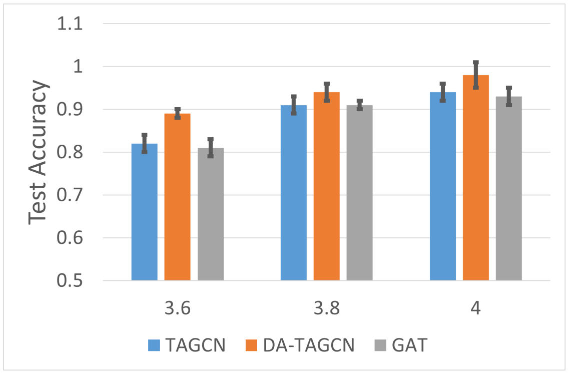
Fig. 5.19 shows graph attention network (GAT) and degree-aware TAGCN (DA-TAGCN) compared to TAGCN for 3 types of synthetic datasets vs. differences in model generation parameters that change the average degree, e.g., for smallworld, it is the mean degree κ.

In general, for all plots, DA-TAGCN performs the best (or one of of the best) as it has weights that train on the degree of the nodes. TAGCN performs similarly to GAT for Erdős-Rényi and preferential attachment graphs (Fig. 5.19a, 5.19c. Fig. 5.19b shows that for smallworld, TAGCN performs better than GCN. This shows that degree of polynomial filters is reasonable at capturing the difference in degree for smallworld graphs. On the other hand, GAT learns the edge weights using the graph signals that are not distinguishable (all have means of 0).

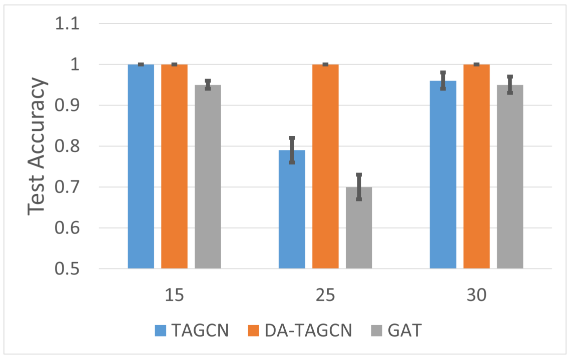
5.4.3.6 Metrics vs GNN

Fig. 5.20 shows the results of classifiers based on network metrics or signal statistics vs GNN (TAGCN). Since we generate the class labels synthetically, it is simple to reverse engineer using the generated graphs $\mathbf{A}$ or the signals $\mathbf{X}$ to find the optimal classifiers, which are either network metrics or signal statistics. In general, these classifiers based on data metrics can outperform TAGCN since TAGCN does not directly capture this information. Instead information is usually not captured at any stage or lost at the aggregation stage (which is required to generate the same length feature vectors to compare graphs of different sizes and orders).

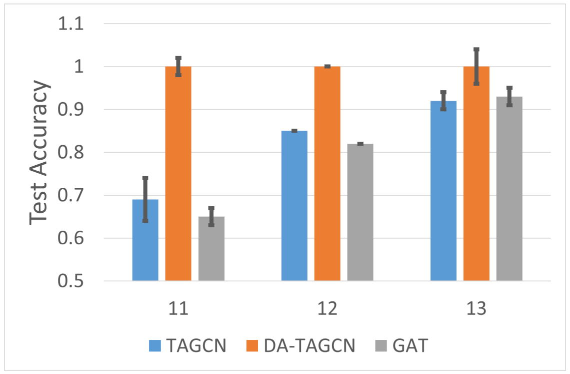
Fig. 5.20a shows the results of classifiers with class differences generated using different r of Erdős Rényi models. As class 1's edge creation probability r_1 approaches class 0's r_0, all models approach random accuracy (50%). As expected, both TAGCN and av-



(a) Erdős-Rényi class 1 edge creation probability r_1 (class 0's r_0 is fixed at 3)



(b) Smallworld class 1 mean degree κ_1 (class 0's κ_0 is fixed at 20)



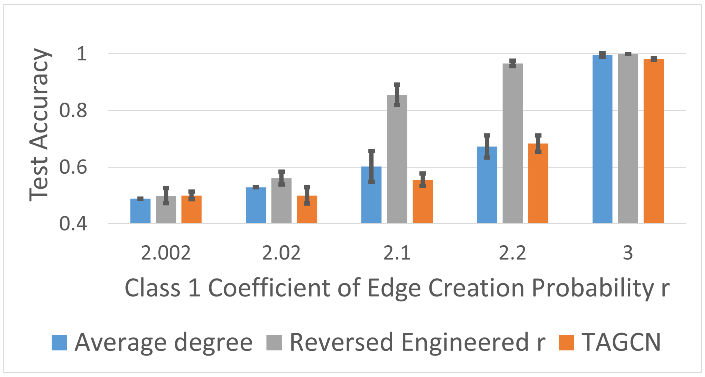
(c) Preferential attachment class 1 edges-to-attach m_1 (class 0's m_0 is fixed at 10)

Figure 5.19: TAGCN vs TAGCN Variants (DA-TAGCN) for 3 synthetic graph generation models. Subcaption indicates the model generation parameter plotted on the x-axis. Colored bars represent average test accuracy of each architecture. Vertical black lines on the bars represent standard deviation.

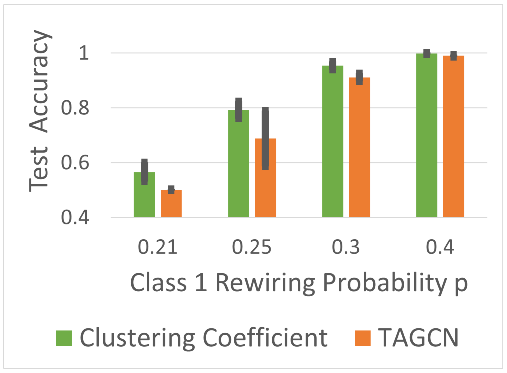
erage degree are worse than the reverse engineered r_1 (calculated using average degree divided by $\ln N$). However, the neural network is able to perform quite well when the difference in ratio is large enough.

Fig. 5.20b shows the results of classifiers with class labels defined using different rewiring probability p for smallworld models. In this case, using either the average degree or reversed engineered r fail (random accuracy of 50%), as they are the same for all graphs. Clustering cofficient (and some other graph clustering metrics like number of triangles) leads to better performance than TAGCN when the rewiring probabilities between classes are close, and similarly when they are different.

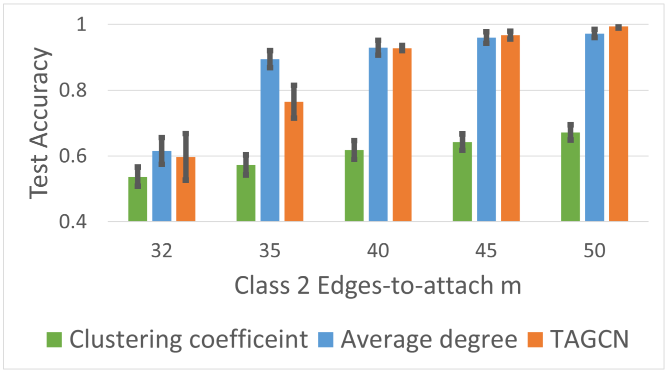
Fig. 5.20c shows the results of classifiers with class labels defined using different edges-to-attach of m_1 of preferential attachment models. TAGCN performs slightly bet-



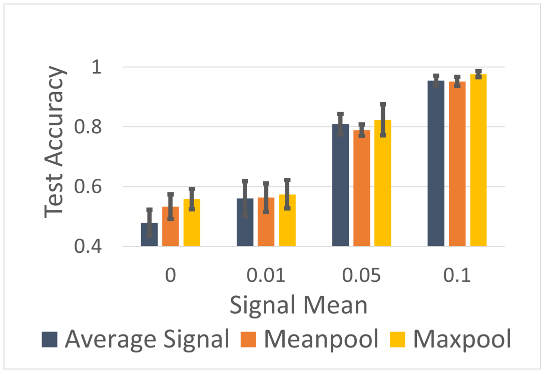
(a) Erdős Rényi: Class 0's r is fixed at 2.



(b) Smallworld: Class 0's edge rewiring probability p is fixed at 0.2.



(c) Preferential Attachment: Class 0's edges-to-attach m is fixed at 30.



(d) Erdős Rényi: Signal mean.

Figure 5.20: TAGCN vs network metrics for 3 synthetic graph generation models. Colored bars represent average test accuracy of each metric/architecture. Vertical black lines on the bars represent standard deviation.

ter than average degree when the differences between the two classes' edges-to-attach are large, and worse when the differences are small. Clustering coefficient fails here.

Fig. 5.20d shows the results of classifiers with class labels defined using difference in signal means with Erdős Rényi models. TAGCN performs similar to using just average signal as the feature regardless of the difference in signal means. Max aggregation (selecting the maximum node value after several convolutions) is better than default mean aggregation.

AND and XOR We consider cases when the graph signals and graph structures are complementary and contrasting. Contrasting case (XOR) is shown in Table 5.1, complementary case (AND) is when graph structure type B and graph signal type B is changed to class 1 in the same table. We apply the default Erdős-Rényi setup for class 0 and

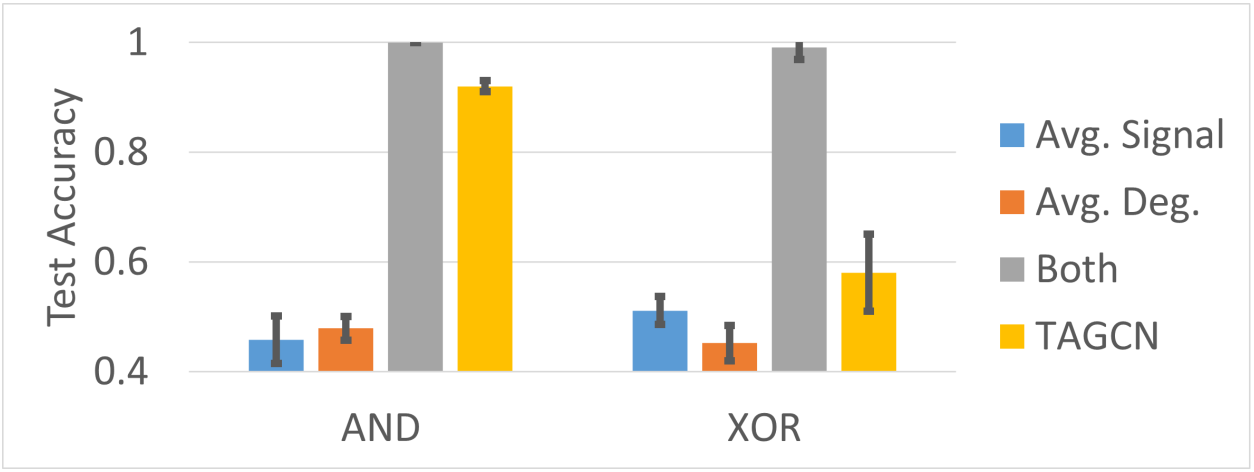


Figure 5.21: TAGCN vs metrics for AND & XOR cases. Colored bars represent average test accuracy of each metric/architecture. Vertical black lines on the bars represent standard deviation.

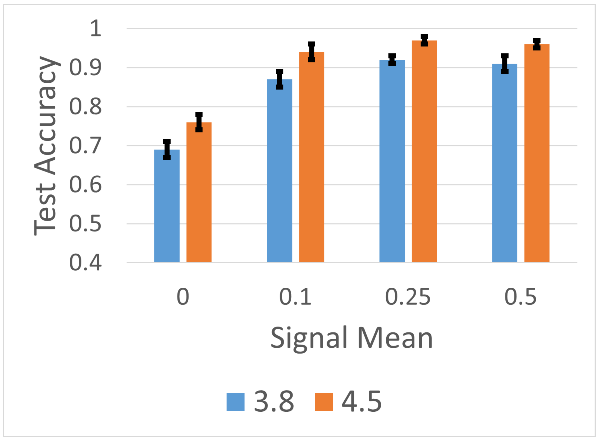
class 1.

Figure 5.21 shows the results of using the simple metrics vs TAGCN for these 2 cases. Using a simple classifier (linear-SVC) with just the average signal or the average degree performs random, and using both performs close to 100%. TAGCN performs well for the AND case and poorly for the XOR case. Default graph convolutions cannot separate the XOR. There are many simple (and complicated) solutions to this. For example, we can simply include the metrics as inputs alongside the output of TAGCN (using ensemble learning). A more complicated approach would be to combine a sumpool TAGCN (defined in Section 5.3.3.2) and DA-TAGCN (defined in Section 5.3.3.3) before linear layer (one network is ideal for capturing the difference in signal means and the other is ideal for capturing the difference in degree).

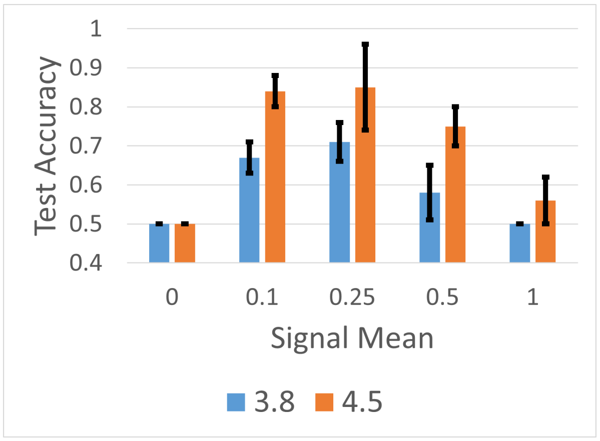
Fig. 5.22 shows TAGCN performance for different graph structures and graph signals. As shown in Fig. 5.22a, for the AND case, as class 1 and class 0 become more different in terms of either the signal or the graph, the performance improves. As shown in Fig. 5.22b, for the XOR case, there seems to be an optimal signal mean for both graph structures (around 0.25). Below and above that the performance drops to random.

5.5 Conclusions

In conclusion, we have shown how the graph structure affects the performance of GNNs for the graph classification task on real and synthetic data. In general, for the two main



(a) AND



(b) XOR

Figure 5.22: TAGCN AND and XOR cases vs different different Erdős-Rényi model generation parameters. Colored bars represent average test accuracy of edge creation probability for class 1 r_1 ($r_0 = 3.0$), and x-axis is difference in signal mean. Vertical black lines on the bars represent standard deviation.

GNN architectures that we studied (GCN and TAGCN), not many layers are needed to achieve optimal performance (compared with computer vision and natural language processing). TAGCN requires fewer number of layers than GCN, with moderate degree of the polynomial filters. Unlike with node classification, graph structure plays a more important role than the graph signal. GNNs can often classify graphs using just the graph structures of the different classes if they are distinct enough. For synthetic datasets, Erdős-Rényi and preferential attachment models have similar test accuracy curves for both GCN and TAGCN. For smallworld model, more than 1 layer is needed to achieve good performance if the edge rewiring probability is distinct for different classes. GNNs are also not ideal for capturing certain network parameters (compared to simple network metrics).

For real datasets, we relate simple network metrics and signal statistics on real datasets to the performance of these models. We show for biological and social network datasets, classifiers on number of edges, number of nodes, or signal means can lead to better or similar performance as graph neural networks. Based on these observations, we were able to apply simple modifications to GCN and TAGCN to improve their performance (sumpool and degree-aware TAGCN).

For real datasets, we also look into the impact of the GNN architecture on its performance. In general, datasets with high average degree and low average diameter require less number of layers to achieve optimal performance. Over-smoothing also occurs for graph classification. It happens at lower number of layers if the graphs have high average diameter though there are some exceptions, depending on the graph signal. TAGCN tends to perform better with higher degrees of the polynomial filters at low number of layers on datasets with low average degree.

For synthetic datasets, we relate the performance to different graph characteristics and GNN architecture. In general, there is no significant difference between increasing class 1 average degree (while keeping class 0 fixed) using Erdős-Rényi's edge creation probability vs preferential attachment's edges-to-attach. Both GCN and TAGCN respond similarly to both of these graph characteristics. For smallworld, we find that it is the interclass difference in edge rewiring probability that leads to the drop in accuracy at layer 1. If they are the same for both classes, then GCN and TAGCN performs well at just 1 layer. This suggests that Reddit-Binary dataset, which also requires at least 2 layers to perform well, may have smallworld characteristics with a high interclass difference in edge rewiring probability.

We also find that simple network metrics that capture different interclass graph characteristics can perform better than GNN (e.g., clustering coefficient performs better than TAGCN for capturing the smallworld's edge rewiring probability). This suggests that GNNs are not optimal for capturing Erdős-Rényi's edge creation probability. Just like real datasets, we also show that simple modification (degree-aware TAGCN) can improve the performance by capturing interclass difference in degree. However, another popular variant (graph attention network) does not perform better than TAGCN on capturing these graph characteristics. Finally, we show that while TAGCN can perform well when graph signal and graph structure have an AND relationship, it fails when they have an XOR relationship.

Chapter 6: Node Classification Training Convergence

The most common algorithm for training a neural network model is gradient-based optimization using backpropagation. Graph neural networks (GNNs) are no different. However, unlike linear and convolutional layers, the optimization of graph convolutional layers is less studied. Theoretically, given the non-convexity of GNN, it is still an open question under what conditions will training loss converges to global minimum. This motivates the question we address here: what properties of the TAGCN architecture and data graph affect the speed of convergence in training.

We theoretically show that training loss converges to the global minimum for linearized TAGCN of any number of layers and degree of the polynomial filters. In general, convergence is faster with more depth and higher degree. The experiments confirm that our theoretical results hold for real datasets, with and without nonlinearity. To be clear, faster training convergence does not imply better test accuracy. As we have shown in Chapter 4, test accuracy drops significantly when the neural network architecture is too large due to over-smoothing.

There are several related works in this area. The first set of related works are on theoretical analysis of linearized neural networks, in terms of their loss landscape [88, 89], global convergence [90, 91], and neural tangent kernel [92]. The second set of related work are on learning dynamics of GNNs, including graph neural tangent kernel [93, 94], and stability analysis [95]. Our work most closely follow [96, 97, 98]. Reference [96] proves global convergence for one-layer GNN, [97] studies GNN optimization with skip connections and greater depth, and specialized initialization, and [98] analyzes the convergence of one-layer GNNs with ReLU nonlinearity.

We organize the chapter as follows. We start by formally defining the problem of training convergence in Section 6.1. We then lay out the theory for different cases in

Section 6.2, before diving into the full proof for the 3 cases: H-layer 1-degree, 1-layer K-degree, and H-layer K-degree in Sections 6.2.1, 6.2.2, and 6.2.3, respectively. We validate the results experimentally in Section 6.3. Finally, we summarize the results in Section 6.4.

6.1 Problem Definition

Following the same notations as Section 4.1, we are given a graph $\mathbf{G}$ and labels of a subset of the nodes $\mathcal{I} \subseteq [N]$, the task is to predict the labels of the remaining nodes $[N] \setminus \mathcal{I}$. The graph $\mathbf{G}$ consists of a set of nodes V with $|V| = N$, signal $\mathbf{X} \in \mathbb{R}^{N \times C}$ on the nodes (where C is the number of features on each node), a set of edges $\mathcal{E}$, and an adjacency matrix $\mathbf{A}$. $\mathbf{A}_{ij} = 0$ unless there is an edge $e = (i, j)$ connecting node i and node j.

The goal is to learn a neural network function $f(\mathbf{X}, \mathbf{A})$ such that the prediction error between f and the true label $\mathbf{Y}$ is small. We use a non-negative real-valued loss function to model the prediction error:

$$L(f(\mathbf{X}, \mathbf{A}), \mathbf{Y}). \tag{6.1}$$

For GNNs, the function is a set of convolutional layers. For TAGCN, each convolutional layer is:

$$\mathbf{X}^{(\ell+1)} = \sigma\left(\sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}^{(\ell)} \mathbf{W}_k^{(\ell)}\right), \tag{3.2}$$

where $\mathbf{W}_k^{(\ell)} \in \mathbb{R}^{C_\ell \times C_{\ell+1}}$, $\mathbf{X}^{(\ell)} \in \mathbb{R}^{N \times C_\ell}$, and the input signal $\mathbf{X} = \mathbf{X}^{(0)}$. We prove training convergence when there is no nonlinearity σ. Then for TAGCN, each convolutional layer is:

$$\mathbf{X}^{(\ell+1)} = \sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}^{(\ell)} \mathbf{W}_k^{(\ell)}, \tag{6.2}$$

We start with the case when there is no polynomial filter and there are no residual connections:

$$\mathbf{X}^{(\ell+1)} = \mathbf{A}^k \mathbf{X}^{(\ell)} \mathbf{W}_k^{(\ell)}. \tag{6.3}$$

The neural network function is then:

$$f(\mathbf{X}, \mathbf{A}, \mathbf{W}, \mathbf{B}) = \mathbf{X}^{(H)} \mathbf{B} \tag{6.4}$$

where $\mathbf{B} \in \mathbb{R}^{C_H \times C_y}$ is the weight matrix for converting to C_y, the number of output channels of $\mathbf{Y}$ (for multi-label classification). We can absorb $\mathbf{B}$ into $\mathbf{W}$, but for this case we include it to make the derivation clearer. The global minimum loss, for which we want the network to converges to is:

Definition 3. (Global minimum) L^* is the global minimum value of GNN f

$$L^* = \inf_{\mathbf{W},\mathbf{B}} L(f(\mathbf{X}, \mathbf{A}, \mathbf{W}, \mathbf{B}), \mathbf{Y}), \tag{6.5}$$

where, for node classification, the loss is over all the nodes in training.

The goal is to prove the convergence:

$$\lim_{t\to\infty} L(\mathbf{W}_t, \mathbf{B}_t) - L^* = 0, \tag{6.6}$$

and to study the rate of convergence in terms of the number of layers and degrees of the polynomial filters.

6.2 Convergence Theory

In this section, we present the main results on the global convergence for TAGCN for node classification. We follow previous related work on this topic [97, 99, 100] and extend them to the TAGCN architecture. We divide the proof into these steps: 1) H-layer, 1-degree, 2) 1-layer, K-degree, 3) H-layer K-degree, where layer refers to the number of convolutions, and degree refers to the order of the polynomial filters.

In the first case, we consider linearized TAGCN with no residual connection (as shown in (6.3)), so it functions similarly to GCN.

Theorem 2. *Let f be an H-layer linear 1-degree TAGCN and loss*

$$L(\widehat{\mathbf{Y}}, \mathbf{Y}) = \|\widehat{\mathbf{Y}} - \mathbf{Y}\|_F^2 \tag{6.7}$$

where $\widehat{\mathbf{Y}}, \mathbf{Y} \in \mathbb{R}^{N\times C_y}$. Then, for any time step $T > 0$,

$$L(\mathbf{W}_T) - L_H^* \le (L(\mathbf{W}_0) - L_H^*) e^{-4\lambda_T^{(H)} \sigma^2_{\min}((\mathbf{A}^H\mathbf{X})_{*\mathcal{I}})T}, \tag{6.8}$$

where $*\mathcal{I}$ selects the nodes in training, $\sigma_{min}(\cdot)$ is the smallest singular value, and

$$\lambda_T^{(H)} := \inf_{t\in[0,T]} \lambda_{\min}(\mathbf{W}_t{}^{(0)}\mathbf{W}_t{}^{(1)} \cdots \mathbf{W}_t{}^{(H-1)}(\mathbf{W}_t{}^{(0)}\mathbf{W}_t{}^{(1)} \cdots \mathbf{W}_t{}^{(H-1)})^\top)$$

where $\lambda_{\min}$ selects the smallest eigenvalue.

As T approaches infinity, the exponent approaches 0, and hence the loss approaches the global minimum. This is guaranteed if $\sigma^2_{\min}((\mathbf{A}^H\mathbf{X})_{*\mathcal{I}}) > 0$ and $\lambda_T^{(H)} > 0$. The first term depends on the data graph signal and structure, and it is true for popular benchmark datasets. The second term is guaranteed under a good initialization, i.e., if the weights are initialized satisfying this condition (since the loss decreases monotonically). See Section 6.2.1 for the complete proof.

Next we consider the case of 1-layer K-degree TAGCN.

Theorem 3. *Let f be a 1-layer K-degree linear TAGCN and loss*

$$L(\widehat{\mathbf{Y}}, \mathbf{Y}) = \|\widehat{\mathbf{Y}} - \mathbf{Y}\|_F^2 \tag{6.9}$$

where $\widehat{\mathbf{Y}}, \mathbf{Y} \in \mathbb{R}^{N \times C_y}$. Let

$$\mathbf{G}_K := [\mathbf{X}, (\mathbf{A}\mathbf{X}), \ldots, (\mathbf{A}^K\mathbf{X})]^\top \in \mathbb{R}^{(K+1)C_0} \times N. \tag{6.10}$$

Then for any time step $T > 0$,

$$L(\mathbf{W}_T, \mathbf{B}_T) - L_H^* \le (L(\mathbf{W}_0, \mathbf{B}_0) - L_H^*)e^{-4\sigma^2_{\min}((\mathbf{G}_K)_{*\mathcal{I}})T} \tag{6.11}$$

As T approaches infinity, the exponent approaches 0, and hence the loss approaches the global minimum. This is guaranteed if $\sigma^2_{\min}((\mathbf{G}_K)_{*\mathcal{I}}) > 0$. This again is true for popular benchmark datasets. Note that good initialization is not required here. See Section 6.2.2 for the complete proof.

Finally, we consider multilayer multidegree case. A H-layer K-degree is the same as 1-layer, HK-degree, since there is no nonlinearity.

Theorem 4. *Let f be an H-layer K-degree linear TAGCN and loss $L(\widehat{\mathbf{Y}}, \mathbf{Y}) = \|\widehat{\mathbf{Y}} - \mathbf{Y}\|_F^2$ where $\widehat{\mathbf{Y}}, \mathbf{Y} \in \mathbb{R}^{N \times C_y}$. Let*

$$\mathbf{G}_{KH} := [\mathbf{X}, (\mathbf{A}\mathbf{X}), \ldots, (\mathbf{A}^{KH}\mathbf{X})]^\top \in \mathbb{R}^{(KH+1)C_0} \times N. \tag{6.12}$$

Then for any time step $T > 0$,

$$L(\mathbf{W}_T, \mathbf{B}_T) - L_H^* \le (L(\mathbf{W}_0, \mathbf{B}_0) - L_H^*)e^{-4\sigma^2_{\min}((\mathbf{G}_{KH})_{*\mathcal{I}})T} \tag{6.13}$$

As T approaches infinity, the exponent approaches 0 at a faster rate, and hence the loss approaches the global minimum. This is guaranteed if $\sigma^2_{\min}((\mathbf{G}_{KH})_{*\mathcal{I}}) > 0$. See Section 6.2.3 for the complete proof.

6.2.1 H-Layer, 1-Degree Convergence Proof

In this section, we prove Theorem 2, the training convergence of H-layer, 1-degree TAGCN. We first compute the gradients of the loss with respect to the parameter $\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{W})$ in Section 6.2.1.1. Using these relations, we then analyze the dynamics induced in $\mathbf{W}^{(0)}\mathbf{W}^{(1)}\cdots\mathbf{W}^{(H-1)}$ in Section 6.2.1.2, and the dynamics induced in the loss value $L(\mathbf{W})$ in Section 6.2.1.3. Finally, we complete the proof using square loss in Section 6.2.1.4.

Despite lack of non-linear activations σ, the loss $L(f(\mathbf{X},\mathbf{A},\mathbf{W},\mathbf{B})$ for TAGCNs (and GCNs) is non-convex. To find the global convergence, we analyze the learning process via gradient flow, i.e., by taking infinitesimal steps during gradient descent. In each t step, the network weights evolve as:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \alpha \nabla_{\mathbf{W}x_t} L \tag{6.14}$$

So, the change in $\mathbf{W}$ with respect to t is:

$$\frac{d}{dt}\mathbf{W}_t = -\frac{\partial L}{\partial \mathbf{W}}(\mathbf{W}_t), \tag{6.15}$$

And similarly for $\mathbf{B}_t$

6.2.1.1 Derivation of $\nabla_B L(B, W)$ and $\nabla_{W^{(\ell)}} L(B, W)$

We relate the gradients $\nabla_{\mathbf{W}^{(\ell)}} L$ to the gradient $\nabla_{(H)} L$, which we define as

$$\nabla_{(H)} L(\mathbf{B},\mathbf{W}) := ((\mathbf{A}^H\mathbf{X})_{*\mathcal{I}})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \hat{\mathbf{Y}}} \in \mathbb{R}^{C_0 \times C_y}. \tag{6.16}$$

Using the relation between $(\nabla_{\mathbf{B}} L, \nabla_{\mathbf{W}^{(\ell)}} L)$ and $\nabla_{(H)} L$, we analyze the dynamics induced in the space of $\mathbf{W}^{(1)}\mathbf{W}^{(2)}\cdots\mathbf{W}^{(\ell)}$ and the space of the loss value $L(\mathbf{B},\mathbf{W})$ in Section 6.2.1.2, and Section 6.2.1.3, respectively. Finally, we complete the proof with square loss in Section 6.2.1.4.

We first prove the relationship of the gradients between $\nabla_{\mathbf{B}} L$ and $\nabla_{\mathbf{W}^{(\ell)}} L$ vs $\nabla_{(H)} L$ in the following lemma:

Lemma 1. *Let f be an H-layer linear TAGCN. Then for any $(\mathbf{B},\mathbf{W})$,*

$$\nabla_{\mathbf{B}} L(\mathbf{B},\mathbf{W}) = (\mathbf{W}^{(0)}\mathbf{W}^{(1)}\cdots\mathbf{W}^{(H-1)})^\top \nabla_{(H)} L(\mathbf{B},\mathbf{W}) \in \mathbb{R}^{C_H \times C_y}, \tag{6.17}$$

and

$$\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{B},\mathbf{W}) = (\mathbf{W}^{(0)}\mathbf{W}^{(1)}\cdots\mathbf{W}^{(\ell-2)}\mathbf{W}^{(\ell-1)})^\top \tag{6.18}$$

$$\nabla_{(H)} L(\mathbf{B},\mathbf{W})(\mathbf{W}^{(\ell+1)}\cdots\mathbf{W}^{(H-1)}\mathbf{B})^\top \in \mathbb{R}^{C_l \times C_{l+1}}. \tag{6.19}$$

Proof of Lemma 1. From (6.4) and (6.3), we have:

$$\widehat{\mathbf{Y}} = f(\mathbf{X}, \mathbf{A}, \mathbf{W}, \mathbf{B})_{*\mathcal{I}} = (\mathbf{X}^{(H)}\mathbf{B})_{*\mathcal{I}}$$

and

$$\mathbf{X}^{(\ell)} = \mathbf{A}\mathbf{X}^{(\ell-1)}\mathbf{W}^{(\ell-1)},$$

(ignoring the nonlinearity and the residual connection). We derive $\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{B}]} \in \mathbb{R}^{C_y N \times C_y C_H}$:

$$\begin{aligned}\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{B}]} &= \frac{\partial}{\partial \operatorname{vec}[\mathbf{B}]} \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}\mathbf{B}] \\ &= \frac{\partial}{\partial \operatorname{vec}[\mathbf{B}]} [I_{C_y} \otimes (\mathbf{X}^{(H)})_{*\mathcal{I}}] \operatorname{vec}[\mathbf{B}] \\ &= [I_{C_y} \otimes (\mathbf{X}^{(H)})_{*\mathcal{I}}] \in \mathbb{R}^{C_y N \times C_y C_y} \end{aligned} \tag{6.20}$$

With this, we derive the formula for $\nabla_{\mathbf{B}} L(\mathbf{B}, \mathbf{W}) \in \mathbb{R}^{C_H \times C_y}$:

$$\frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \operatorname{vec}[\mathbf{B}]} = \frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{B}]} = \frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} [I_{C_y} \otimes (\mathbf{X}^{(H)})_{*\mathcal{I}}]$$

Thus,

$$\begin{aligned}\nabla_{\operatorname{vec}[\mathbf{B}]} L(\mathbf{B}, \mathbf{W}) &= \left(\frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \operatorname{vec}[\mathbf{B}]}\right)^{\top} \\ &= [I_{C_y} \otimes (\mathbf{X}^{(H)})_{*\mathcal{I}}^{\top}] \left(\frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}\right)^{\top} \\ &= [I_{C_y} \otimes (\mathbf{X}^{(H)})_{*\mathcal{I}}^{\top}] \operatorname{vec}\left[\frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \widehat{\mathbf{Y}}}\right] \\ &= \operatorname{vec}\left[(\mathbf{X}^{(H)})_{*\mathcal{I}}^{\top} \frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \widehat{\mathbf{Y}}}\right] \in \mathbb{R}^{C_H C_Y}.\end{aligned}$$

Therefore,

$$\nabla_{\mathbf{B}} L(\mathbf{B}, \mathbf{W}) = (\mathbf{X}^{(H)})_{*\mathcal{I}}^{\top} \frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \widehat{\mathbf{Y}}} \in \mathbb{R}^{C_H \times C_Y}. \tag{6.21}$$

Next, we derive $\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \in \mathbb{R}^{C_y N \times C_l C_{l+1}}$:

$$\begin{aligned}\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} &= \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}\mathbf{B}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \\ &= \frac{\partial}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} [\mathbf{B}^{\top} \otimes I_N] \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}]]\end{aligned}$$

$$
\begin{aligned}
&= [\mathbf{B}^\top \otimes I_N] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}]]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \\
&= [\mathbf{B}^\top \otimes I_N] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} \frac{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \\
&= [\mathbf{B}^\top \otimes I_N] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} \frac{\partial \operatorname{vec}[\mathbf{A}\mathbf{X}^{(\ell)}\mathbf{W}^{(\ell)}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \\
&= [\mathbf{B}^\top \otimes I_N] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} \frac{\partial [I_{C_l} \otimes \mathbf{A}\mathbf{X}^{(\ell)}] \operatorname{vec}[\mathbf{W}^{(\ell)}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \\
&= [\mathbf{B}^\top \otimes I_N] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} [I_{C_l} \otimes \mathbf{A}\mathbf{X}^{(\ell)}] \qquad (6.22)
\end{aligned}
$$

From (6.3), we have:

$$
\mathbf{X}^{(\ell)} = \mathbf{A}\mathbf{X}^{(\ell-1)}\mathbf{W}^{(\ell-1)} \tag{6.23}
$$

so

$$
\operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}] = \operatorname{vec}[\mathbf{A}_{*\mathcal{I}}\mathbf{X}^{(H-1)}\mathbf{W}^{(H-1)}] = (\mathbf{W}^{(H-1)^\top} \otimes \mathbf{A}_{*\mathcal{I}}) \operatorname{vec}[\mathbf{X}^{(H-1)}] \tag{6.24}
$$

and

$$
\operatorname{vec}[(\mathbf{X}^{(\ell)})] = \operatorname{vec}[\mathbf{A}\mathbf{X}^{(\ell-1)}\mathbf{W}^{(\ell-1)}] = (\mathbf{W}^{(\ell-1)^\top} \otimes \mathbf{A}) \operatorname{vec}[\mathbf{X}^{(\ell-1)}]. \tag{6.25}
$$

By recursively applying (6.25), we get

$$
\begin{aligned}
\operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}] &= (\mathbf{W}^{(H-1)^\top} \otimes \mathbf{A}_{*\mathcal{I}})(\mathbf{W}^{(H-2)^\top} \otimes \mathbf{A}) \cdots (\mathbf{W}^{(\ell+1)^\top} \otimes \mathbf{A}) \operatorname{vec}[\mathbf{X}^{(\ell+1)}] \\
&= (\mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \otimes (\mathbf{A}^{H-l-1})_{*\mathcal{I}}) \operatorname{vec}[\mathbf{X}^{(\ell+1)}].
\end{aligned}
$$

Hence,

$$
\frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} = \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \otimes (\mathbf{A}^{H-l-1})_{*\mathcal{I}} \tag{6.26}
$$

Combining (6.22) and (6.26) gives:

$$
\begin{aligned}
\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} &= [\mathbf{B}^\top \otimes I_N] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} [I_{C_l} \otimes \mathbf{A}\mathbf{X}^{(\ell)}] \\
&= [\mathbf{B}^\top \otimes I_N][\mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \otimes (\mathbf{A}^{H-l-1})_{*\mathcal{I}}][I_{C_l} \otimes \mathbf{A}\mathbf{X}^{(\ell)}] \\
&= \mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \otimes (\mathbf{A}^{H-l})_{*\mathcal{I}}\mathbf{X}^{(\ell)} \in \mathbb{R}^{C_y N \times C_l C_{l+1}}. \qquad (6.27)
\end{aligned}
$$

With this, we derive the formula of $\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{B}, \mathbf{W}) \in \mathbb{R}^{C_l \times C_{l+1}}$:

$$\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} = \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} = \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} [\mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \otimes (\mathbf{A}^{H-l})_{*\mathcal{I}} \mathbf{X}^{(\ell)}].$$

Thus, with $\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \in \mathbb{R}^{N \times C_y}$,

$$\begin{aligned}
\nabla_{\operatorname{vec}[\mathbf{W}^{(\ell)}]} L(\mathbf{B},\mathbf{W}) &= [\mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \otimes (\mathbf{A}^{H-l})_{*\mathcal{I}} \mathbf{X}^{(\ell)}]^\top \left(\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \right)^\top \\
&= [(\mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top})^\top \otimes \mathbf{X}^{(\ell)^\top} (\mathbf{A}^{H-l})_{*\mathcal{I}}^\top] \left(\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \right)^\top \\
&= [(\mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top})^\top \otimes \mathbf{X}^{(\ell)^\top} (\mathbf{A}^{H-l})_{*\mathcal{I}}^\top] \operatorname{vec}\left[\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \right] \\
&= \operatorname{vec}\left[\mathbf{X}^{(\ell)^\top} (\mathbf{A}^{H-l})_{*\mathcal{I}}^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \right] \in \mathbb{R}^{C_l C_{l+1}} \\
&= \operatorname{vec}\left[((\mathbf{A}^{H-l})_{*\mathcal{I}} \mathbf{X}^{(\ell)})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \right] \in \mathbb{R}^{C_l C_{l+1}}
\end{aligned}$$

Therefore,

$$\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{B},\mathbf{W}) = ((\mathbf{A}^{H-l})_{*\mathcal{I}} \mathbf{X}^{(\ell)})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \in \mathbb{R}^{C_l \times C_{l+1}}. \quad (6.28)$$

Now, we can prove the statements of this lemma with the following notation:

$$\nabla_{(\ell)} L(\mathbf{B},\mathbf{W}) := (\mathbf{A}^l \mathbf{X})_{*\mathcal{I}}^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \in \mathbb{R}^{C_0 \times C_y}. \quad (6.29)$$

Plugging this into (6.21):

$$\begin{aligned}
\nabla_{\mathbf{B}} L(\mathbf{B},\mathbf{W}) &= (\mathbf{X}^{(H)})_{*\mathcal{I}}^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \\
&= (\mathbf{A}_{*\mathcal{I}} \mathbf{X}^{(H-1)} \mathbf{W}^{(H-1)})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \\
&= (\mathbf{A}_{*\mathcal{I}}^H \mathbf{X} \mathbf{W}^{(0)} \cdots \mathbf{W}^{(H-1)})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \\
&= (\mathbf{W}^{(0)} \cdots \mathbf{W}^{(H-1)})^\top ((\mathbf{A}^H)_{*\mathcal{I}} \mathbf{X})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \\
&= (\mathbf{W}^{(0)} \cdots \mathbf{W}^{(H-1)})^\top \nabla_{(H)} L(\mathbf{B},\mathbf{W})
\end{aligned}$$

Using (6.29) along with (6.28):

$$\begin{aligned}
\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{B},\mathbf{W}) &= (\mathbf{A}_{*\mathcal{I}}^{H-l}\mathbf{X}^{(\ell)})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \\
&= (\mathbf{A}_{*\mathcal{I}}^{H-l+l}\mathbf{X}\mathbf{W}^{(0)} \cdots \mathbf{W}^{(\ell-1)})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \\
&= (\mathbf{W}^{(0)} \cdots \mathbf{W}^{(\ell-2)}\mathbf{W}^{(\ell-1)})^\top (\mathbf{A}_{*\mathcal{I}}^{H}\mathbf{X})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \\
&= (\mathbf{W}^{(0)} \cdots \mathbf{W}^{(\ell-2)}\mathbf{W}^{(\ell-1)})^\top \nabla_{(H)} L(\mathbf{B},\mathbf{W}) \mathbf{B}^\top \mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \\
&= (\mathbf{W}^{(0)} \cdots \mathbf{W}^{(\ell-2)}\mathbf{W}^{(\ell-1)})^\top \nabla_{(H)} L(\mathbf{B},\mathbf{W}) (\mathbf{W}^{(\ell+1)} \cdots \mathbf{W}^{(H-1)}\mathbf{B})^\top
\end{aligned}$$

□

6.2.1.2 Dynamics Induced in the Space of $\mathbf{W}^{(0)}\mathbf{W}^{(2)} \cdots \mathbf{W}^{(\ell)}\mathbf{B}$

The discrete cases of gradient descent:

$$\mathbf{W}^{(\ell)'} = \mathbf{W}^{(\ell)} - \alpha \nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{B},\mathbf{W})$$

$$\mathbf{B}' = \mathbf{B} - \alpha \nabla_{\mathbf{B}} L(\mathbf{B},\mathbf{W})$$

The iterative application of the dynamics:

$$\mathbf{W}^{(0)'} \cdots \mathbf{W}^{(\ell)'}\mathbf{B}' = (\mathbf{W}^{(0)} - \alpha \nabla_{\mathbf{W}^{(0)}} L(\mathbf{B},\mathbf{W})) \cdots (\mathbf{W}^{(\ell)} - \alpha \nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{B},\mathbf{W}))(\mathbf{B} - \alpha \nabla_{\mathbf{B}} L(\mathbf{B},\mathbf{W}))$$

For ease of notation, we define:

$$\mathbf{Z}^{(H)} := \mathbf{W}^{(0)} \cdots \mathbf{W}^{(H-1)}\mathbf{B}$$

and

$$\mathbf{Z}^{(H)'} := \mathbf{W}^{(0)'} \cdots \mathbf{W}^{(H-1)'}\mathbf{B}'$$

So,

$$\begin{aligned}
\mathbf{Z}^{(H)'} &= \mathbf{W}^{(0)'} \cdots \mathbf{W}^{(\ell)'}\mathbf{B}' \\
&= (\mathbf{W}^{(0)} - \alpha \nabla_{\mathbf{W}^{(0)}} L(\mathbf{B},\mathbf{W})) \cdots (\mathbf{W}^{(H-1)} - \alpha \nabla_{\mathbf{W}^{(H-1)}} L(\mathbf{B},\mathbf{W}))(\mathbf{B} - \alpha \nabla_{\mathbf{B}} L(\mathbf{B},\mathbf{W}))
\end{aligned}$$

Expanding the multiplication, we get:

$$\mathbf{Z}^{(H)'} = \mathbf{Z}^{(H)} - \alpha \mathbf{W}^{(0)}\mathbf{W}^{(1)} \cdots \mathbf{W}^{(H-1)} \nabla_{\mathbf{B}} L(\mathbf{B},\mathbf{W})$$

$$- \alpha \sum_{i=0}^{H-1} \mathbf{W}^{(0)} \dots \mathbf{W}^{(i-1)} \nabla_{\mathbf{W}^{(i)}} L(\mathbf{B}, \mathbf{W})) \mathbf{W}^{(i+1)} \dots \mathbf{W}^{(H-1)} \mathbf{B} + O(\alpha^2)$$

Vectorizing both sides:

$$\begin{aligned}
&\operatorname{vec}[\mathbf{Z}^{(H)'}] - \operatorname{vec}[\mathbf{Z}^{(H)}] \\
&\qquad = -\alpha \operatorname{vec}[\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(H-1)} \nabla_{\mathbf{B}} L(\mathbf{B}, \mathbf{W})] \\
&\qquad - \alpha \sum_{i=0}^{H-1} \operatorname{vec}[\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(i-1)} \nabla_{\mathbf{W}^{(i)}} L(\mathbf{B}, \mathbf{W}) \mathbf{W}^{(i+1)} \dots \mathbf{W}^{(H-1)} \mathbf{B}] \\
&+ O(\alpha^2)
\end{aligned}$$

Now we can plug in (6.17) and (6.18) from Lemma 1 for $\nabla_{\mathbf{B}} L(\mathbf{B}, \mathbf{W})$ and $\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{B}, \mathbf{W})$, respectively. For the first term we get:

$$\begin{aligned}
&\operatorname{vec}[\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(H-1)} \nabla_{\mathbf{B}} L(\mathbf{B}, \mathbf{W})] \\
&\qquad = \operatorname{vec}[\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(H-1)} (\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(H-1)})^\top \nabla_{(H)} L(\mathbf{B}, \mathbf{W})] \\
&\qquad = [I_{C_y} \otimes \mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(H-1)} (\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(H-1)})^\top] \operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})],
\end{aligned}$$

and for the 2nd term:

$$\begin{aligned}
&\sum_{i=0}^{H-1} \operatorname{vec}[\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(i-1)} \nabla_{\mathbf{W}^{(i)}} L(\mathbf{B}, \mathbf{W}) \mathbf{W}^{(i+1)} \dots \mathbf{W}^{(H-1)} \mathbf{B}] \\
&= \sum_{i=0}^{H-1} \operatorname{vec}[\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(i-1)} (\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(i-2)}\mathbf{W}^{(i-1)})^\top \nabla_{(H)} L(\mathbf{B}, \mathbf{W}) \\
&\qquad (\mathbf{W}^{(i+1)} \dots \mathbf{W}^{(H-1)} \mathbf{B})^\top \mathbf{W}^{(i+1)} \dots \mathbf{W}^{(H-1)} \mathbf{B}] \\
&= \sum_{i=0}^{H-1} [\mathbf{B}^\top (\mathbf{W}^{(i+1)} \dots \mathbf{W}^{(H-1)})^\top (\mathbf{W}^{(i+1)} \dots \mathbf{W}^{(H-1)} \mathbf{B}) \\
&\qquad \otimes \mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(i-1)} (\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(i-2)}\mathbf{W}^{(i-1)})^\top] \operatorname{vec}\left[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})\right].
\end{aligned}$$

Combining the two:

$$\begin{aligned}
&\operatorname{vec}[\mathbf{Z}^{(H)'}] - \operatorname{vec}[\mathbf{Z}^{(H)}] \\
&\quad = -\alpha [I_{C_y} \otimes \mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(H-1)} (\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(H-1)} \mathbf{B})^\top] \operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})] \\
&\quad - \alpha \sum_{i=0}^{H-1} [\mathbf{B}^\top (\mathbf{W}^{(i+1)} \dots \mathbf{W}^{(H-1)})^\top (\mathbf{W}^{(i+1)} \dots \mathbf{W}^{(H-1)}) \mathbf{B} \\
&\quad \otimes \mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(i-1)} (\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(i-2)}\mathbf{W}^{(i-1)})^\top] \operatorname{vec}\left[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})\right] + O(\alpha^2)
\end{aligned}$$

Hence, the induced dynamics of $\mathrm{vec}[\mathbf{Z}^{(H)}]$ is

$$\frac{d}{dt}\mathrm{vec}[\mathbf{Z}^{(H)}] = -F_{(H)}\,\mathrm{vec}[\nabla_{(H)}L(\mathbf{B},\mathbf{W})] - \left(\sum_{i=0}^{H-1} J_{(i,H)}^{\top}J_{(i,H)}\right)\mathrm{vec}\left[\nabla_{(H)}L(\mathbf{B},\mathbf{W})\right],$$

where

$$F_{(H)} = [I_{C_y} \otimes \mathbf{W}^{(0)}\mathbf{W}^{(1)}\cdots\mathbf{W}^{(H-1)}(\mathbf{W}^{(0)}\mathbf{W}^{(1)}\cdots\mathbf{W}^{(H-1)})^{\top}],$$

and

$$J_{(i,H)} = [\mathbf{W}^{(i+1)}\cdots\mathbf{W}^{(H-1)}\mathbf{B} \otimes (\mathbf{W}^{(0)}\mathbf{W}^{(1)}\cdots\mathbf{W}^{(i-2)}\mathbf{W}^{(i-1)})^{\top}].$$

6.2.1.3 Dynamics Induced in the Space of Loss $L(\mathbf{B},\mathbf{W})$

Next, we analyze the dynamics induced in the space of the loss $L(\mathbf{B},\mathbf{W})$ using the dynamic of $\mathbf{Z}^{(H)}$ derived in the previous section.

$$\begin{aligned}\frac{d}{dt}L(\mathbf{B},\mathbf{W}) &= \frac{d}{dt}L_0(\mathbf{Z}^{(H)})\\ &= \frac{\partial L_0(\mathbf{Z}^{(H)})}{\partial\,\mathrm{vec}[\mathbf{Z}^{(H)}]}\frac{d\,\mathrm{vec}[\mathbf{Z}^{(H)}]}{dt},\end{aligned}$$

where

$$L_0(\mathbf{Z}^{(H)}) = L(f_0(\mathbf{X},\mathbf{Z}^{(H)})_{*\mathcal{I}},\mathbf{Y}),\;\; f_0(\mathbf{X},\mathbf{Z}^{(H)}) = \mathbf{A}^H\mathbf{X}\mathbf{Z}^{(H)},\;\; \text{and } \mathbf{Z}^{(H)} = \mathbf{W}^{(0)}\cdots\mathbf{W}^{(H-1)}\mathbf{B}.$$

Since

$$f_0(\mathbf{X},\mathbf{Z}^{(H)}) = f(\mathbf{X},\mathbf{B},\mathbf{W}) = \widehat{\mathbf{Y}}$$

and

$$L_0(\mathbf{Z}^{(H)}) = L(\mathbf{B},\mathbf{W}),$$

we have

$$\begin{aligned}\left(\frac{\partial L_0(\mathbf{Z}^{(H)})}{\partial\,\mathrm{vec}[\mathbf{Z}^{(H)}]}\right)^{\top} &= \left(\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial\,\mathrm{vec}[\widehat{\mathbf{Y}}]}\frac{\partial\,\mathrm{vec}[\widehat{\mathbf{Y}}]}{\partial\,\mathrm{vec}[\mathbf{Z}^{(H)}]}\right)^{\top}\\ &= \left(\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial\,\mathrm{vec}[\widehat{\mathbf{Y}}]}\frac{\partial\,\mathrm{vec}[\mathbf{A}^H\mathbf{X}\mathbf{Z}^{(H)}]}{\partial\,\mathrm{vec}[\mathbf{Z}^{(H)}]}\right)^{\top}\\ &= \left(\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial\,\mathrm{vec}[\widehat{\mathbf{Y}}]}\left(\frac{\partial}{\partial\,\mathrm{vec}[\mathbf{Z}^{(H)}]}[I_{C_y}\otimes\mathbf{A}^H\mathbf{X}_{*\mathcal{I}}]\,\mathrm{vec}[\mathbf{Z}^{(H)}]\right)\right)^{\top}\\ &= [I_{C_y}\otimes(\mathbf{A}^H\mathbf{X}_{*\mathcal{I}})^{\top}]\,\mathrm{vec}\left[\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial\widehat{\mathbf{Y}}}\right]\end{aligned}$$

$$
\begin{aligned}
&= \mathrm{vec}[(\mathbf{A}^H\mathbf{X})_{*\mathcal{I}}^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}}] \\
&= \mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})]
\end{aligned}
$$

Combining these,

$$
\begin{aligned}
&\frac{d}{dt}L(\mathbf{B},\mathbf{W}) \\
&= \frac{\partial L_0(\mathbf{Z}^{(H)})}{\partial \,\mathrm{vec}[\mathbf{Z}^{(H)}]} \frac{d\,\mathrm{vec}[\mathbf{Z}^{(H)}]}{dt} \\
&= \mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})]^\top \frac{d\,\mathrm{vec}[\mathbf{Z}^{(H)}]}{dt} \\
&= -\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})]^\top F_{(H)}\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})] \\
&\qquad\qquad - \sum_{i=0}^{H-1} \mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})]^\top J_{(i,H)}^\top J_{(i,H)}\,\mathrm{vec}\left[\nabla_{(H)} L(\mathbf{B},\mathbf{W})\right] \\
&= -\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})]^\top F_{(H)}\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})] - \sum_{i=0}^{H-1} \| J_{(i,H)}\,\mathrm{vec}\left[\nabla_{(H)} L(\mathbf{B},\mathbf{W})\right] \|_2^2
\end{aligned}
$$

Hence,

$$
\begin{aligned}
\frac{d}{dt}L(\mathbf{B},\mathbf{W}) = &-\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})]^\top F_{(H)}\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})] && (6.30)\\
&- \sum_{i=0}^{H-1} \left\| J_{(i,H)}\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})] \right\|_2^2 && (6.31)
\end{aligned}
$$

$F_{(H)}$ is real symmetric and positive semidefinite since it is the Kronecker of the identity and $\mathbf{W}^{(0:H-1)}\mathbf{W}^{(0:H-1)^\top}$, so

$$
\frac{d}{dt}L(\mathbf{B},\mathbf{W}) \le -\lambda_{\min}(F_{(H)}) \| \mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})] \|_2^2 - \sum_{i=0}^{H-1} \left\| J_{(i,H)}\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})] \right\|_2^2.
$$

With $\lambda_{\mathbf{B},\mathbf{W}} = \lambda_{\min}(F_{(H)})$,

$$
\frac{d}{dt}L(\mathbf{B},\mathbf{W}) \le -\lambda_{\mathbf{B},\mathbf{W}} \| \mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})] \|_2^2 - \sum_{i=0}^{H-1} \left\| J_{(i,H)}\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})] \right\|_2^2 \tag{6.32}
$$

6.2.1.4 Completing the Proof of Theorem 2 with Square Loss

Now we complete the proof of theorem 2 with squared loss. It is possible to carry out the proof with other losses, but the derivation is more tedious. So,

$$
L(\mathbf{B},\mathbf{W}) = L(f(\mathbf{X},\mathbf{B},\mathbf{W})_{*\mathcal{I}},\mathbf{Y}) = \| f(\mathbf{X},\mathbf{B},\mathbf{W})_{*\mathcal{I}} - \mathbf{Y} \|_F^2
$$

with $\widehat{\mathbf{Y}} = f(\mathbf{X}, \mathbf{B}, \mathbf{W})_{*\mathcal{I}}$, we have

$$\frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \widehat{\mathbf{Y}}} = \frac{\partial}{\partial \widehat{\mathbf{Y}}} \|\widehat{\mathbf{Y}} - \mathbf{Y}\|_F^2 = 2(\widehat{\mathbf{Y}} - \mathbf{Y}) \in \mathbb{R}^{N \times C_y},$$

and

$$\begin{aligned} \operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})] = \operatorname{vec}\left[(\mathbf{A}^H \mathbf{X}_{*\mathcal{I}})^\top \frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \widehat{\mathbf{Y}}}\right] &= 2 \operatorname{vec}\left[(\mathbf{A}^H \mathbf{X}_{*\mathcal{I}})^\top (\widehat{\mathbf{Y}} - \mathbf{Y})\right] \\ &= 2[I_{C_y} \otimes (\mathbf{A}^H \mathbf{X}_{*\mathcal{I}})^\top] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]. \end{aligned}$$

Hence,

$$\|\operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})]\|_2^2 = 4 \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top [I_{C_y} \otimes (\mathbf{A}^H \mathbf{X}_{*\mathcal{I}})(\mathbf{A}^H \mathbf{X}_{*\mathcal{I}})^\top] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \tag{6.33}$$

Plugging in (6.33) into (6.32),

$$\begin{aligned} \frac{d}{dt} L(\mathbf{B}, \mathbf{W}) &\le -\lambda_{\mathbf{B},\mathbf{W}} \|\operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})]\|_2^2 - \sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})] \right\|_2^2 \\ &= -4\lambda_{\mathbf{B},\mathbf{W}} \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top [I_{C_y} \otimes (\mathbf{A}^H \mathbf{X}_{*\mathcal{I}})(\mathbf{A}^H \mathbf{X}_{*\mathcal{I}})^\top] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \\ &\quad - \sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})] \right\|_2^2 \\ &= -4\lambda_{\mathbf{B},\mathbf{W}} \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top \left[I_{C_y} \otimes \mathbf{G}_H \mathbf{G}_H^\top\right] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] && (6.34) \\ &\quad - \sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})] \right\|_2^2 && (6.35) \end{aligned}$$

where $\mathbf{G}_H := (\mathbf{A}^H)\mathbf{X}_{*\mathcal{I}} \in \mathbb{R}^{N \times C_0}$

Here we apply orthogonal decomposition by projecting $\operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]$ onto the column space of $I_{C_y} \otimes \mathbf{G}_H \in \mathbb{R}^{C_y N \times C_y C_0}$ with

$$\mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H} \in \mathbb{R}^{C_y N \times C_y N}.$$

So the decomposition is

$$\operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] = v + v^\perp,$$

where

$$v = \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H} \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}],$$

and

$$v^\perp = (I_{C_y N} - \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H}) \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}].$$

Hence, plugging into the first term of (6.35):

$$\begin{aligned}
\operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top & \left[I_{C_y} \otimes \mathbf{G}_H \mathbf{G}_H^\top\right] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \\
&= (v + v^\perp)^\top \left[I_{C_y} \otimes \mathbf{G}_H\right] \left[I_{C_y} \otimes \mathbf{G}_H^\top\right] (v + v^\perp) \\
&= v^\top \left[I_{C_y} \otimes \mathbf{G}_H\right] \left[I_{C_y} \otimes \mathbf{G}_H^\top\right] v \\
&\geq \sigma^2_{\min}(\mathbf{G}_H) \| \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H} \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \|_2^2 \\
&= \sigma^2_{\min}(\mathbf{G}_H) \| \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H} \operatorname{vec}[\widehat{\mathbf{Y}}] - \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H} \operatorname{vec}[\mathbf{Y}] \|_2^2 \\
&= \sigma^2_{\min}(\mathbf{G}_H) \| \operatorname{vec}[\widehat{\mathbf{Y}}] - \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H} \operatorname{vec}[\mathbf{Y}] \pm \operatorname{vec}[\mathbf{Y}] \|_2^2 \\
&= \sigma^2_{\min}(\mathbf{G}_H) \| \operatorname{vec}[\widehat{\mathbf{Y}}] - \operatorname{vec}[\mathbf{Y}] + (I_{C_y N} - \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H}) \operatorname{vec}[\mathbf{Y}] \|_2^2 \\
&\geq \sigma^2_{\min}(\mathbf{G}_H) (\| \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \|_2 - \| (I_{C_y N} - \mathbf{P}_{\mathbf{G}_H^\top \otimes I_{C_y}}) \operatorname{vec}[\mathbf{Y}] \|_2)^2 \\
&\geq \sigma^2_{\min}(\mathbf{G}_H) (\| \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \|_2^2 - \| (I_{C_y N} - \mathbf{P}_{\mathbf{G}_H^\top \otimes I_{C_y}}) \operatorname{vec}[\mathbf{Y}] \|_2^2),
\end{aligned}$$

where we use the property that the singular values of the Kronecker product $\left[I_{C_y} \otimes \mathbf{G}_H\right]$ are products of the singular values of I_{C_y} and $\mathbf{G}_H$. Since

$$L(\mathbf{B}, \mathbf{W}) = \| \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \|_2^2$$

and

$$L^*_H = \| (I_{C_y N} - \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H}) \operatorname{vec}[\mathbf{Y}] \|_2^2,$$

$$\operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top \left[I_{C_y} \otimes \mathbf{G}_H \mathbf{G}_H^\top\right] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \geq \sigma^2_{\min}(\mathbf{G}_H)(L(\mathbf{B}, \mathbf{W}) - L^*_H).$$

Therefore,

$$\begin{aligned}
\frac{d}{dt} L(\mathbf{B}, \mathbf{W}) &\leq -4\lambda_{\mathbf{B},\mathbf{W}} \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top \left[I_{C_y} \otimes \mathbf{G}_H \mathbf{G}_H^\top\right] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] - \sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})] \right\|_2^2 \\
&\leq -4\lambda_{\mathbf{B},\mathbf{W}} \sigma^2_{\min}(\mathbf{G}_H)(L(\mathbf{B}, \mathbf{W}) - L^*_H) - \sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})] \right\|_2^2
\end{aligned}$$

Since L^*_H is the global minimum, $\frac{d}{dt} L^*_H = 0$,

$$\frac{d}{dt}(L(\mathbf{B}, \mathbf{W}) - L^*_H) \leq -4\lambda_{\mathbf{B},\mathbf{W}} \sigma^2_{\min}(\mathbf{G}_H)(L(\mathbf{B}, \mathbf{W}) - L^*_H) - \sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})] \right\|_2^2$$

Let $\mathbf{L} = L(\mathbf{B}, \mathbf{W}) - L^*_H$,

$$\frac{d\mathbf{L}}{dt} \leq -4\lambda_{\mathbf{B},\mathbf{W}} \sigma^2_{\min}(\mathbf{G}_H) \mathbf{L} - \sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)} L(\mathbf{B}, \mathbf{W})] \right\|_2^2 \tag{6.36}$$

Both terms are always non-negative, so this derivative is always positive or zero. The starting loss will always be higher than the global minimum $\mathbf{L} \geq 0$. If the loss reaches the global minimum, then the derivative will be 0. Hence we label $\bar{t}$ as the point where it reaches the global minimum. Then within $[0, \bar{t}]$, equation (6.36) implies that

$$\frac{1}{\mathbf{L}}\frac{d\mathbf{L}}{dt} \leq -4\lambda_{\mathbf{B},\mathbf{W}}\sigma^2_{\min}(\mathbf{G}_H) - \frac{1}{\mathbf{L}}\sum_{i=0}^{H-1}\left\|J_{(i,H)}\operatorname{vec}[\nabla_{(H)}L(\mathbf{B},\mathbf{W})]\right\|_2^2$$

Taking integral over time,

$$\int_0^T \frac{1}{\mathbf{L}}\frac{d\mathbf{L}}{dt}dt \leq -\int_0^T 4\lambda_{\mathbf{B},\mathbf{W}}\sigma^2_{\min}(\mathbf{G}_H)dt - \int_0^T \frac{1}{\mathbf{L}}\sum_{i=0}^{H-1}\left\|J_{(i,H)}\operatorname{vec}[\nabla_{(H)}L(\mathbf{B},\mathbf{W})]\right\|_2^2 dt$$

Applying the substitution rule,

$$\int_0^T \frac{1}{\mathbf{L}}\frac{d\mathbf{L}}{dt}dt = \int_{\mathbf{L}_0}^{\mathbf{L}_T}\frac{1}{\mathbf{L}}d\mathbf{L} = \log(\mathbf{L}_T) - \log(\mathbf{L}_0), \tag{6.37}$$

where

$$\mathbf{L}_0 = L(\mathbf{W}_0, \mathbf{B}_0) - L^* \tag{6.38}$$

and

$$Lb_T = L(\mathbf{W}_T, \mathbf{B}_T) - L^*_H. \tag{6.39}$$

Hence,

$$\log(\mathbf{L}_T) - \log(\mathbf{L}_0) \leq -4\sigma^2_{\min}(\mathbf{G}_H)\int_0^T \lambda_{\mathbf{B},\mathbf{W}}dt - \int_0^T \frac{1}{\mathbf{L}}\sum_{i=0}^{H-1}\left\|J_{(i,H)}\operatorname{vec}[\nabla_{(H)}L(\mathbf{B},\mathbf{W})]\right\|_2^2 dt$$

which implies that

$$\begin{aligned}\mathbf{L}_T &\leq e^{\log(\mathbf{L}_0) - 4\sigma^2_{\min}(\mathbf{G}_H)\int_0^T \lambda_{\mathbf{B},\mathbf{W}}dt - \int_0^T \frac{1}{\mathbf{L}}\sum_{i=0}^{H-1}\left\|J_{(i,H)}\operatorname{vec}[\nabla_{(H)}L(\mathbf{B},\mathbf{W})]\right\|_2^2 dt} \\ &= \mathbf{L}_0 e^{-4\sigma^2_{\min}(\mathbf{G}_H)\int_0^T \lambda_{\mathbf{B},\mathbf{W}}dt - \int_0^T \frac{1}{\mathbf{L}}\sum_{i=0}^{H-1}\left\|J_{(i,H)}\operatorname{vec}[\nabla_{(H)}L(\mathbf{B},\mathbf{W})]\right\|_2^2 dt}\end{aligned}$$

Plugging $\mathbf{L} = L(\mathbf{B}, \mathbf{W}) - L^*_H$ back in, we get

$$L(\mathbf{W}_T, \mathbf{B}_T) - L^*_H \leq (L(\mathbf{W}_0, \mathbf{B}_0) - L^*_H) \tag{6.40}$$

$$e^{-4\sigma^2_{\min}(\mathbf{G}_H)\int_0^T \lambda_{\mathbf{W}_t,\mathbf{B}_t}dt - \int_0^T \frac{1}{L(\mathbf{W}_t,\mathbf{B}_t) - L^*_H}\sum_{i=0}^{H-1}\left\|J_{(i,H)}\operatorname{vec}[\nabla_{(H)}L(\mathbf{W}_t,\mathbf{B}_t)]\right\|_2^2 dt}. \tag{6.41}$$

Using again the property of the Kronecker product,

$$\lambda_{\min}([(\mathbf{B}_{(H),t}\dots\mathbf{B}_{(1),t})^\top\mathbf{B}_{(H),t}\cdots\mathbf{B}_{(1),t}\otimes I_{C_y}]) = \lambda_{\min}((\mathbf{B}_{(H),t}\dots\mathbf{B}_{(1),t})^\top\mathbf{B}_{(H),t}\cdots\mathbf{B}_{(1),t}),$$

that implies that

$$\lambda_T^{(H)} = \inf_{t\in[0,T]} \lambda_{\mathbf{W}_t,\mathbf{B}_t}.$$

Since the 2nd term of the exponential is always greater than 0, i.e.,

$$\int_0^T \frac{1}{L(\mathbf{W}_t,\mathbf{B}_t) - L_H^*} \sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)} L(\mathbf{W}_t,\mathbf{B}_t)] \right\|_2^2 dt \geq 0$$

we get

$$\begin{aligned} L(\mathbf{W}_T,\mathbf{B}_T) - L_H^* &\leq (L(\mathbf{W}_0,\mathbf{B}_0) - L_H^*) \\ &\qquad e^{-4\lambda_T^{(H)}\sigma^2_{\min}(\mathbf{G}_H)T - \int_0^T \frac{1}{L(\mathbf{W}_t,\mathbf{B}_t)-L_H^*}\sum_{i=0}^{H-1}\|J_{(i,H)}\operatorname{vec}[\nabla_{(H)}L(\mathbf{W}_t,\mathbf{B}_t)]\|_2^2 dt} \\ &\leq (L(\mathbf{W}_0,\mathbf{B}_0) - L_H^*) e^{-4\lambda_T^{(H)}\sigma^2_{\min}(\mathbf{G}_H)T} \\ &= (L(\mathbf{W}_0,\mathbf{B}_0) - L_H^*) e^{-4\lambda_T^{(H)}\sigma^2_{\min}(\mathbf{A}^H\mathbf{X}_{*\mathcal{I}})T} \end{aligned}$$

Here, we arrive at Theorem 2 and the proof is complete. □

6.2.2 1-Layer, K-Degree Convergence Proof

In this section, we prove Theorem 3, the training convergence of 1-layer, K-degree TAGCN. This follows closely from the previous section. For the sake of completeness, we include all the details. Just like the previous section, we first compute the gradients of the loss with respect to the parameter $\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{W})$ in Section 6.2.2.1. Using these relations, we then analyze the dynamics induced in $\mathbf{W}^{(\ell)}$ in Section 6.2.2.2, and the dynamics induced in the loss value $L(\mathbf{W})$ in Section 6.2.2.3. Finally, we complete the proof using square loss in Section 6.2.2.4.

6.2.2.1 Derivation of $\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{W})$

We first prove the relationship of the gradients $\nabla_{\mathbf{B}} L$, $\nabla_{\mathbf{W}_j} L$ and $\nabla_{(H)} L$ in the following lemma:

Lemma 2. *Let f be an 1-layer K-degree linear TAGCN. Then for any $(\mathbf{W})$,*

$$\nabla_{\mathbf{W}_j} L(\mathbf{W}) = \nabla_j L(\mathbf{W}) \tag{6.42}$$

where

$$\nabla_j L(\mathbf{W}) := (\mathbf{A}^j\mathbf{X})_{*\mathcal{I}}^\top \frac{\partial L(\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \in \mathbb{R}^{C_0\times C_y} \tag{6.43}$$

Proof of Lemma 2. From (6.2), we have

$$\mathbf{X}^{(H)} = \sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}^{(0)} \mathbf{W}_k$$

(ignoring the nonlinearity). For simplicity, we use $\mathbf{W}_k = \mathbf{W}_k{}^{(0)}$ since there is only one layer. We also remove $\mathbf{B}$ weights, so

$$\widehat{\mathbf{Y}} = f(\mathbf{X}, \mathbf{A}, \mathbf{W})_{*\mathcal{I}} = (\mathbf{X}^{(H)})_{*\mathcal{I}}. \tag{6.44}$$

We derive $\nabla_{\mathbf{W}_j} L(\mathbf{W})$, starting with $\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}_j]} \in \mathbb{R}^{C_y N \times C_l C_{l+1}}$:

$$\begin{aligned}
\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}_j]} &= \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}]}{\partial \operatorname{vec}[\mathbf{W}_j]} \\
&= \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})_{*\mathcal{I}}]]}{\partial \operatorname{vec}[\mathbf{W}_j]} \\
&= \frac{\partial \operatorname{vec}[\sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}^{(0)} \mathbf{W}_k]}{\partial \operatorname{vec}[\mathbf{W}_j]} \\
&= \frac{\partial [I_{C_y} \otimes \mathbf{A}^j \mathbf{X}^{(0)}] \operatorname{vec}[\mathbf{W}_j]}{\partial \operatorname{vec}[\mathbf{W}_j]} \\
&= [I_{C_y} \otimes \mathbf{A}^j \mathbf{X}^{(0)}]
\end{aligned} \tag{6.45}$$

Hence, the formula of $\nabla_{\mathbf{W}_j} L(\mathbf{W}) \in \mathbb{R}^{C_0 \times C_y}$:

$$\frac{\partial L(\mathbf{W})}{\partial \operatorname{vec}[\mathbf{W}_j]} = \frac{\partial L(\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}_j]} = \frac{\partial L(\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} [I_{C_y} \otimes \mathbf{A}^j \mathbf{X}^{(0)}]$$

Thus, with $\frac{\partial L(\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \in \mathbb{R}^{N \times C_y}$,

$$\nabla_{\operatorname{vec}[\mathbf{W}_j]} L(\mathbf{W}) = \operatorname{vec}[(\mathbf{A}^j \mathbf{X})^\top \frac{\partial L(\mathbf{W})}{\partial \widehat{\mathbf{Y}}}] \in \mathbb{R}^{C_0 C_y}$$

Therefore,

$$\nabla_{\mathbf{W}_j} L(\mathbf{W}) = (\mathbf{A}^j \mathbf{X})^\top \frac{\partial L(\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \in \mathbb{R}^{C_0 \times C_y}. \tag{6.46}$$

Now, we can prove the statements of this lemma with the (6.43):

$$\begin{aligned}
\nabla_{\mathbf{W}_j} L(\mathbf{W}) &= \operatorname{vec}[(\mathbf{A}^j \mathbf{X})^\top \frac{\partial L(\mathbf{W})}{\partial \widehat{\mathbf{Y}}}] \\
&= \nabla_j L(\mathbf{W})
\end{aligned}$$

□

Using Lemma 2, we prove Theorem 3.

6.2.2.2 Dynamics Induced in the Space of $\mathbf{W}^{(\ell)}$

Similar to the previous section, the dynamics are as follows:

$$\mathbf{W}_j' = \mathbf{W}_j' - \alpha \nabla_{\mathbf{W}_j} L(\mathbf{W})$$

Vectorizing both sides:

$$\begin{aligned} \operatorname{vec}[\mathbf{W}_j'] - \operatorname{vec}[\mathbf{W}_j] &= -\alpha \nabla_{\mathbf{W}_j} L(\mathbf{W}) \\ &= -\alpha \operatorname{vec}[\nabla_j L(\mathbf{W})] \end{aligned}$$

using Lemma 2 for $\nabla_{\mathbf{W}_j} L(\mathbf{W})$.

Hence, the induced dynamics of $\operatorname{vec}[\mathbf{W}_j]$ is simply

$$\frac{d}{dt} \operatorname{vec}[\mathbf{W}_j] = -\operatorname{vec}[\nabla_j L(\mathbf{W})]$$

6.2.2.3 Dynamics Induced in the Space of Loss $L(\mathbf{W})$

Next, we analyze the dynamics induced in the space of the loss $L(\mathbf{W})$ using the dynamic of $\mathbf{W}_j$ derived in the previous section. Let

$$L(\mathbf{W}) := L(f(\mathbf{X}, \mathbf{W}), \mathbf{Y})$$

where L is the square loss function (to be analyzed later). Using chain rule,

$$\begin{aligned} \frac{d}{dt} L(\mathbf{W}) &= \frac{d}{dt} L_0(\mathbf{W}_0, \dots, \mathbf{W}_K) \\ &= \sum_{k=0}^{K} \frac{\partial L_0(\mathbf{W}_0, \dots, \mathbf{W}_K)}{\partial \operatorname{vec}[\mathbf{W}_k]} \frac{d \operatorname{vec}[\mathbf{W}_k]}{dt}, \end{aligned}$$

where

$$L_0(\mathbf{W}_0, \dots, \mathbf{W}_K) = L(f_0(\mathbf{X}, \mathbf{W}), \mathbf{Y}), \quad f_0(\mathbf{X}, \mathbf{W}) = \sum_{k=0}^{K} \mathbf{A}^k \mathbf{X} \mathbf{W}_k, \quad \text{and } \mathbf{W}_k = \mathbf{W}_k$$

Since

$$f_0(\mathbf{X}) = f(\mathbf{X}, \mathbf{W}) = \widehat{\mathbf{Y}}$$

and $L_0(\mathbf{W}_0, \dots, \mathbf{W}_K) = L(\mathbf{W})$, we have

$$\left(\frac{\partial L_0(\mathbf{W}_0, \dots, \mathbf{W}_K)}{\partial \operatorname{vec}[\mathbf{W}_k]} \right)^\top = \left(\frac{\partial L(\mathbf{W}, \mathbf{B})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}_k]} \right)^\top$$

$$\begin{aligned}
&= \left(\frac{\partial L(\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}\left(\frac{\partial}{\partial \operatorname{vec}[\mathbf{W}_k]}\operatorname{vec}[\sum_{k=0}^{K}\mathbf{A}^k\mathbf{X}\mathbf{W}_k]\right)\right)^\top \\
&= \left(\frac{\partial L(\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}\left(\frac{\partial}{\partial \operatorname{vec}[\mathbf{W}_k]}\sum_{k=0}^{K}[I_{C_y}\otimes(\mathbf{A}^k\mathbf{X})]\operatorname{vec}[\mathbf{W}_k]\right)\right)^\top \\
&= \left(\frac{\partial L(\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}[I_{C_y}\otimes(\mathbf{A}^k\mathbf{X})]\right)^\top \\
&= [I_{C_y}\otimes(\mathbf{A}^k\mathbf{X})^\top]\operatorname{vec}\left[\frac{\partial L(\mathbf{W})}{\partial\widehat{\mathbf{Y}}}\right] \\
&= \operatorname{vec}\left[(\mathbf{A}^k\mathbf{X})^\top\frac{\partial L(\mathbf{W})}{\partial\widehat{\mathbf{Y}}}\right] \\
&= \operatorname{vec}[\nabla_k L(\mathbf{W})]
\end{aligned}$$

Combining these,

$$\begin{aligned}
\frac{d}{dt}L(\mathbf{W}) &= \sum_{k=0}^{K}\frac{\partial L_0(\mathbf{W}_0,\ldots,\mathbf{W}_K)}{\partial \operatorname{vec}[\mathbf{W}_k]}\frac{d\operatorname{vec}[\mathbf{W}_k]}{dt} \\
&= \sum_{k=0}^{K}\operatorname{vec}[\nabla_k L(\mathbf{W})]^\top\frac{d\operatorname{vec}[\mathbf{W}_k]}{dt} \\
&= -\sum_{k=0}^{K}\operatorname{vec}[\nabla_k L(\mathbf{W})]^\top\operatorname{vec}[\nabla_k L(\mathbf{W})] \qquad (6.47)
\end{aligned}$$

6.2.2.4 Completing the Proof of Theorem 3 with Square Loss

Given that

$$L(\mathbf{W},\mathbf{B}) = \ell(f(\mathbf{X},\mathbf{W}),\mathbf{Y}) = \|f(\mathbf{X},\mathbf{W})-\mathbf{Y}\|_F^2$$

with $\widehat{\mathbf{Y}} = f(\mathbf{X},\mathbf{W})$, we have

$$\frac{\partial L(\mathbf{W})}{\partial\widehat{\mathbf{Y}}} = \frac{\partial}{\partial\widehat{\mathbf{Y}}}\|\widehat{\mathbf{Y}}-\mathbf{Y}\|_F^2 = 2(\widehat{\mathbf{Y}}-\mathbf{Y})\in\mathbb{R}^{N\times C_y},$$

and hence

$$\begin{aligned}
\operatorname{vec}[\nabla_{(k)}L(\mathbf{W})] &= \operatorname{vec}\left[(\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}^\top\frac{\partial L(\mathbf{W})}{\partial\widehat{\mathbf{Y}}}\right] \\
&= 2\operatorname{vec}\left[(\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}^\top(\widehat{\mathbf{Y}}-\mathbf{Y})\right] \\
&= 2[I_{C_y}\otimes(\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}^\top]\operatorname{vec}[\widehat{\mathbf{Y}}-\mathbf{Y}].
\end{aligned}$$

Therefore,

$$\|\operatorname{vec}[\nabla_{(l)}L(\mathbf{W})]\|_2^2 = 4\operatorname{vec}[\widehat{\mathbf{Y}}-\mathbf{Y}]^\top[I_{C_y}\otimes(\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}(\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}^\top]\operatorname{vec}[\widehat{\mathbf{Y}}-\mathbf{Y}]. \qquad (6.48)$$

Now, we can plug in (6.47)

$$\begin{aligned}
\frac{d}{dt}L(\mathbf{W}) &= -\sum_{k=0}^{K} \operatorname{vec}[\nabla_k L(\mathbf{W})]^\top \operatorname{vec}[\nabla_k L(\mathbf{W})] \\
&= -\sum_{k=0}^{K} 4 \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top [I_{C_y} \otimes (\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}(\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}^\top] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \\
&= -4 \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top \left(\sum_{k=0}^{K} I_{C_y} \otimes (\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}(\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}^\top \right) \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \\
&= -4 \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top \left(I_{C_y} \otimes \sum_{k=0}^{K} (\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}(\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}^\top \right) \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \\
&= -4\lambda_{\mathbf{B},\mathbf{W}} \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top \left(I_{C_y} \otimes \mathbf{G}_K\mathbf{G}_K^\top \right) \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]
\end{aligned}$$

where

$$\mathbf{G}_K\mathbf{G}_K^\top = \begin{bmatrix} \mathbf{X}_{*\mathcal{I}} \\ (\mathbf{A}\mathbf{X})_{*\mathcal{I}} \\ \vdots \\ (\mathbf{A}^K\mathbf{X})_{*\mathcal{I}} \end{bmatrix} \begin{bmatrix} \mathbf{X}_{*\mathcal{I}} \\ (\mathbf{A}\mathbf{X})_{*\mathcal{I}} \\ \vdots \\ (\mathbf{A}^K\mathbf{X})_{*\mathcal{I}} \end{bmatrix}^\top = \sum_{l=0}^{H} (\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}(\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}^\top$$

Here we apply orthogonal decomposition by projecting $\operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]$ onto the column space of

$$I_{C_y} \otimes \mathbf{G}_K \in \mathbb{R}^{C_y N \times (K+1) C_y C_0}.$$

So

$$\mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H} \in \mathbb{R}^{C_y N \times C_y N},$$

and the decomposition is

$$\operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] = v + v^\perp,$$

where

$$v = \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_K} \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]$$

and

$$v^\perp = (I_{C_y N} - \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_K}) \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}].$$

Then,

$$\operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top \left[I_{C_y} \otimes \mathbf{G}_K\mathbf{G}_K^\top \right] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] = (v + v^\perp)^\top \left[I_{C_y} \otimes \mathbf{G}_K \right] \left[I_{C_y} \otimes \mathbf{G}_K^\top \right] (v + v^\perp)$$

$$
\begin{aligned}
&= v^\top \left[I_{C_y} \otimes \mathbf{G}_K \right] \left[I_{C_y} \otimes \mathbf{G}_K^\top \right] v \\
&\geq \sigma^2_{\min}(\mathbf{G}_K) \| \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_K} \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \|_2^2 \\
&= \sigma^2_{\min}(\mathbf{G}_K) \| \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_K} \operatorname{vec}[\widehat{\mathbf{Y}}] - \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_K} \operatorname{vec}[\mathbf{Y}] \|_2^2 \\
&= \sigma^2_{\min}(\mathbf{G}_K) \| \operatorname{vec}[\widehat{\mathbf{Y}}] - \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_K} \operatorname{vec}[\mathbf{Y}] \pm \operatorname{vec}[\mathbf{Y}] \|_2^2 \\
&= \sigma^2_{\min}(\mathbf{G}_K) \| \operatorname{vec}[\widehat{\mathbf{Y}}] - \operatorname{vec}[\mathbf{Y}] + (I_{C_y N} - \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_K}) \operatorname{vec}[\mathbf{Y}] \|_2^2 \\
&\geq \sigma^2_{\min}(\mathbf{G}_K) (\| \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \|_2 - \| (I_{C_y N} - \mathbf{P}_{\mathbf{G}_K^\top \otimes I_{C_y}}) \operatorname{vec}[\mathbf{Y}] \|_2)^2 \\
&\geq \sigma^2_{\min}(\mathbf{G}_K) (\| \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \|_2^2 - \| (I_{C_y N} - \mathbf{P}_{\mathbf{G}_K^\top \otimes I_{C_y}}) \operatorname{vec}[\mathbf{Y}] \|_2^2,
\end{aligned}
$$

where we again use the property that the singular values of the Kronecker product $\left[I_{C_y} \otimes \mathbf{G}_K \right]$ are products of the singular values of I_{C_y} and $\mathbf{G}_K$.

Since

$$L(\mathbf{W}) = \| \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \|_2^2$$

and

$$L_H^* = \| (I_{C_y N} - \mathbf{P}_{I_{C_y} \otimes \mathbf{G}_K}) \operatorname{vec}[\mathbf{Y}] \|_2^2$$

, then

$$\operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top \left[I_{C_y} \otimes \mathbf{G}_K \mathbf{G}_K^\top \right] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \geq \sigma^2_{\min}(\mathbf{G}_K)(L(\mathbf{B}, \mathbf{W}) - L_H^*).$$

Therefore,

$$
\begin{aligned}
\frac{d}{dt} L(\mathbf{W}) &\leq -4 \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}]^\top \left[I_{C_y} \otimes \mathbf{G}_K \mathbf{G}_K^\top \right] \operatorname{vec}[\widehat{\mathbf{Y}} - \mathbf{Y}] \\
&\leq -4 \sigma^2_{\min}(\mathbf{G}_K)(L(\mathbf{B}, \mathbf{W}) - L_H^*)
\end{aligned}
$$

Since L_H^* is the global minimum, $\frac{d}{dt} L_H^* = 0$,

$$\frac{d}{dt}(L(\mathbf{B}, \mathbf{W}) - L_H^*) \leq -4 \sigma^2_{\min}(\mathbf{G}_K)(L(\mathbf{B}, \mathbf{W}) - L_H^*)$$

Let $\mathbf{L} = L(\mathbf{B}, \mathbf{W}) - L_H^*$,

$$\frac{d\mathbf{L}}{dt} \leq -4 \sigma^2_{\min}(\mathbf{G}_K) \mathbf{L} \tag{6.49}$$

Since this is always non-negative, so this derivative is always non-positive. The starting loss will always be higher than the global minimum $\mathbf{L} \geq 0$. If the loss reaches the global minimum, then the derivative will be 0. Hence we label $\bar{t}$ as the point where it reaches the global minimum. Then within $[0, \bar{t}]$, equation (6.49) implies that

$$\frac{1}{\mathbf{L}}\frac{d\mathbf{L}}{dt} \leq -4\sigma^2_{\min}(\mathbf{G}_K)$$

Taking integral over time,

$$\int_0^T \frac{1}{\mathbf{L}}\frac{d\mathbf{L}}{dt}dt \leq -\int_0^T 4\sigma^2_{\min}(\mathbf{G}_K)dt = -\sigma^2_{\min}(\mathbf{G}_K)T$$

Applying the substitution rule,

$$\int_0^T \frac{1}{\mathbf{L}}\frac{d\mathbf{L}}{dt}dt = \int_{\mathbf{L}_0}^{\mathbf{L}_T} \frac{1}{\mathbf{L}}d\mathbf{L} = \log(\mathbf{L}_T) - \log(\mathbf{L}_0),$$

where

$$\mathbf{L}_0 = L(\mathbf{W}_0, \mathbf{B}_0) - L^*$$

and

$$\mathbf{L}_T = L(\mathbf{W}_T, \mathbf{B}_T) - L^*_H.$$

Hence,

$$\log(\mathbf{L}_T) - \log(\mathbf{L}_0) \leq -4\sigma^2_{\min}(\mathbf{G}_K)T$$

which implies that

$$\begin{aligned}\mathbf{L}_T &\leq e^{\log(\mathbf{L}_0) - 4\sigma^2_{\min}(\mathbf{G}_K)T} \\ &= \mathbf{L}_0 e^{-4\sigma^2_{\min}(\mathbf{G}_K)T}\end{aligned}$$

Plugging $\mathbf{L} = L(\mathbf{B}, \mathbf{W}) - L^*_H$ back in, we get

$$L(\mathbf{W}_T, \mathbf{B}_T) - L^*_H \leq (L(\mathbf{W}_0, \mathbf{B}_0) - L^*_H)e^{-4\sigma^2_{\min}(\mathbf{G}_K)T} \tag{6.50}$$

Here, we arrive at Theorem 3 and the proof is complete. □

6.2.3 H-Layer, K-Degree Convergence Proof

We show that with variable substitution, we can reduce H-layer K-degree TAGCN to 1-layer, HK-degree (since there is no nonlinearity), and it is also guaranteed to converge like the previous case with HK-degree. However, the actual convergence rate may be different due to different initializations.

From (6.2), we have $\mathbf{X}^{(\ell)} = \sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}^{(\ell-1)} \mathbf{W}_k{}^{(\ell-1)}$. We start with the 2-layer case:

$$\mathbf{X}^{(2)} = \sum_{k_1=0}^{K} \mathbf{A}^{k_1}\mathbf{X}^{(1)}{\mathbf{W}_{k_1}}^{(1)}$$

Plugging in $\mathbf{X}^{(1)} = \sum_{k_0=0}^{K} \mathbf{A}^{k_0}\mathbf{X}^{(0)}{\mathbf{W}_{k_0}}^{(0)}$ gives:

$$\begin{aligned}\mathbf{X}^{(2)} = \sum_{k_1=0}^{K} \mathbf{A}^{k_1}[\sum_{k_0=0}^{K} \mathbf{A}^{k_0}\mathbf{X}^{(0)}{\mathbf{W}_{k_0}}^{(0)}]{\mathbf{W}_{k_1}}^{(1)} \\ \sum_{k_1=0}^{K}\sum_{k_0=0}^{K} \mathbf{A}^{k_0+k_1}\mathbf{X}^{(0)}{\mathbf{W}_{k_0}}^{(0)}{\mathbf{W}_{k_1}}^{(1)}\end{aligned}$$

And $\mathbf{X}^{(3)}$ is:

$$\begin{aligned}\mathbf{X}^{(3)} &= \sum_{k_2=0}^{K} \mathbf{A}^{k_2}\mathbf{X}^{(2)}{\mathbf{W}_{k_2}}^{(2)} \\ &= \sum_{k_2=0}^{K} \mathbf{A}^{k_2}\sum_{k_1=0}^{K}\sum_{k_0=0}^{K} \mathbf{A}^{k_0+k_1}\mathbf{X}^{(0)}{\mathbf{W}_{k_0}}^{(0)}{\mathbf{W}_{k_1}}^{(1)}{\mathbf{W}_{k_2}}^{(2)} \\ &= \sum_{k_2=0}^{K}\sum_{k_1=0}^{K}\sum_{k_0=0}^{K} \mathbf{A}^{k_0+k_1+k_2}\mathbf{X}^{(0)}{\mathbf{W}_{k_0}}^{(0)}{\mathbf{W}_{k_1}}^{(1)}{\mathbf{W}_{k_2}}^{(2)}\end{aligned}$$

By recursively building up, we get:

$$\mathbf{X}^{H} = \sum_{k_{H-1}=0}^{K}\sum_{k_{H-2}=0}^{K}\cdots\sum_{k_0=0}^{K} \mathbf{A}^{k_0+k_1+k_2+\cdots+k_{H-1}}\mathbf{X}^{(0)}{\mathbf{W}_{k_0}}^{(0)}{\mathbf{W}_{k_1}}^{(1)}{\mathbf{W}_{k_2}}^{(2)}\cdots{\mathbf{W}_{k_{H-1}}}^{(H-1)}$$

This is the same as 1-Layer KH-degree case if we combine the coefficients of the same polynomial. For 1-Layer KH:

$$\mathbf{X}^{(H)} = \sum_{i=0}^{KH} \mathbf{A}^{i}\mathbf{X}^{(0)}{\mathbf{W}_{i}}^{(\text{1 layer})},$$

so,

$${\mathbf{W}_0}^{(\text{1 layer})} = {\mathbf{W}_0}^{(0)}{\mathbf{W}_0}^{(1)}\cdots{\mathbf{W}_0}^{(H-1)},$$

and in general,

$${\mathbf{W}_i}^{(\text{1 layer})} = \sum_{k_0,k_1,\cdots,k_{H-1}\in S_k|\sum_{j=0}^{H-1}k_j=i} {\mathbf{W}_{k_1}}^{(1)}{\mathbf{W}_{k_2}}^{(2)}\cdots{\mathbf{W}_{k_{H-1}}}^{(H-1)}$$

where $S_k = (0, 1, \cdots, K)$

Hence, following similar derivation, the results are similar to (6.50) in the previous section, i.e.,

$$L(\mathbf{W}_T, \mathbf{B}_T) - L_H^* \leq (L(\mathbf{W}_0, \mathbf{B}_0) - L_H^*)e^{-4\sigma^2_{\min}(\mathbf{G}_K)T}, \tag{6.50}$$

but with

$$\mathbf{G}_K\mathbf{G}_K^\top = \begin{bmatrix} \mathbf{X} \\ \mathbf{AX} \\ \vdots \\ \mathbf{A}^{KH}\mathbf{X} \end{bmatrix}^\top \begin{bmatrix} \mathbf{X} \\ \mathbf{AX} \\ \vdots \\ \mathbf{A}^{KH}\mathbf{X} \end{bmatrix} = \sum_{l=0}^{H} (\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}(\mathbf{A}^k\mathbf{X})_{*\mathcal{I}}^\top.$$

6.3 Experimental Analysis

We experimentally validate our results on real datasets, using linearized TAGCN, by showing that its train loss decreases monotonically over time to a global minimum, which is 0 in most cases. Higher number of layers and degree of the polynomial filters accelerate learning. We then show that the loss also converges for nonlinearized TAGCN.

6.3.1 Data Descriptions

We study the training convergence of 6 real node classification benchmark datasets (CORA, CiteSeer, PubMed, Amazon Computers, Coauthor Physics, and Coauthor CS). See Section 4.3.1 for their descriptions.

6.3.2 Experimental Setup

We follow a similar experimental setup as described in Section 4.3.2. For the linearized case, we decrease the learning rate of ADAM to 0.0001, remove weight decay, and remove the stopping condition. Hence, we train for 10,000 epochs, when previously, we have applied early stopping condition if the validation loss is not improving. We apply 32 hidden channels, and consider 1-10 number of layers and 1-10 degree of the polynomial filters.

6.3.3 Results

Fig. 6.1 and 6.2 shows the results of train loss for 10,000 epochs for H-layer, 1-degree GCN, and 2-Layer K-degree TAGCN. In general, the train loss decreases monotonically over time at a log-linear rate.

In Fig. 6.1, for Amazon Computer and Coauthor Physics, we see spikes in training losses. They are a consequence of exploding gradients from model instability (the weights grow exponentially). Approaches to fixing this include gradient clipping and weight regularization (which we have for Chapter 4, but removed here). For Amazon Computers, the spikes start occurring at layer 4 (for layer 1-3, there are no spikes).

6.3.3.1 Number of Layers

In Section 6.2.1.3, we showed that the convergence rate for H-layer, 1-degree is:

$$\frac{d}{dt}L(\mathbf{B},\mathbf{W}) = -\operatorname{vec}[\nabla_{(H)}L(\mathbf{B},\mathbf{W})]^\top F_{(H)} \operatorname{vec}[\nabla_{(H)}L(\mathbf{B},\mathbf{W})] \tag{6.30}$$

$$-\sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)}L(\mathbf{B},\mathbf{W})] \right\|_2^2 . \tag{6.51}$$

The second term $\sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)}L(\mathbf{B},\mathbf{W})] \right\|_2^2$ is a summation over positive terms and it increases as H the number of layers increase, thereby accelerating the learning. Fig. 6.1 validates this. Increasing the number of layers accelerates the convergence especially at the earlier epochs.

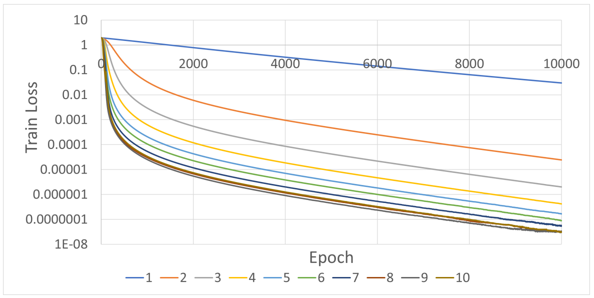
6.3.3.2 Degree of The Polynomial Filters

In Section 6.2.2.3, we showed that the convergence rate for 1-layer, K-degree is:

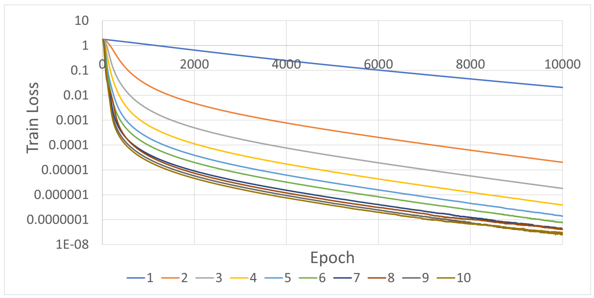
$$\frac{d}{dt}L(\mathbf{W}) = -\sum_{k=0}^{K} \operatorname{vec}[\nabla_k L(\mathbf{W})]^\top \operatorname{vec}[\nabla_k L(\mathbf{W})] \tag{6.47}$$

Again, this is a summation over positive terms and it increases as the degree of the polynomial filters K increase. The convergence rate is faster with higher number of layers vs higher degree of polynomial filters, e.g., TAGCN converges faster with 4 layers + 2 degrees compared to 2 layers + 4 degrees.

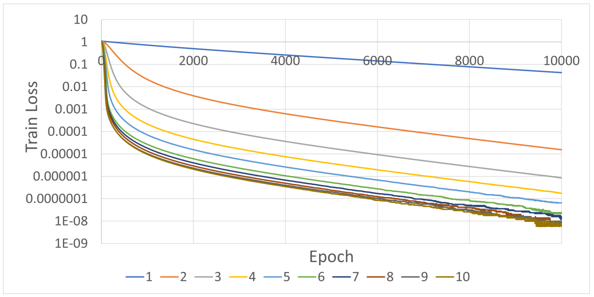
Fig. 6.2 validates this. Increasing the degree of polynomial filters accelerates the convergence. There seems to be no spikes at higher degree compared with higher number of layers. Residual connections allow weights to be updated directly from the loss (instead of propagating through higher degree).



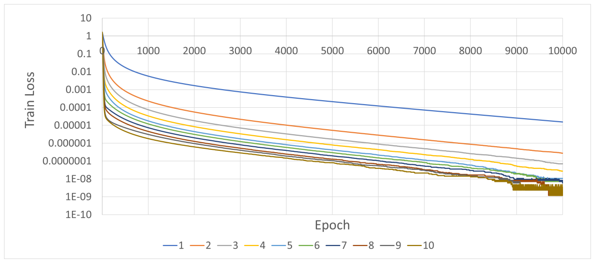
(a) CORA



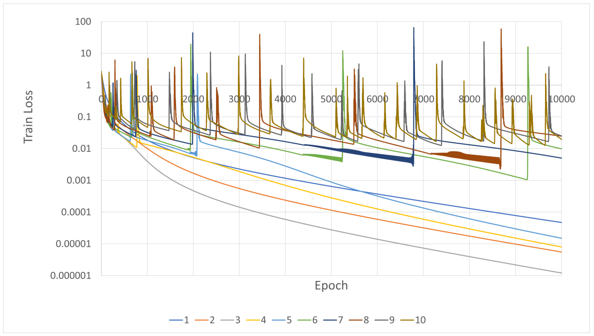
(b) CiteSeer



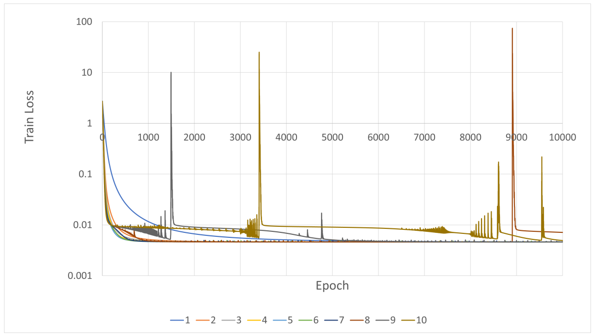
(c) PubMed



(d) Coauthor Physics



(e) Amazon Computer



(f) Coauthor CS

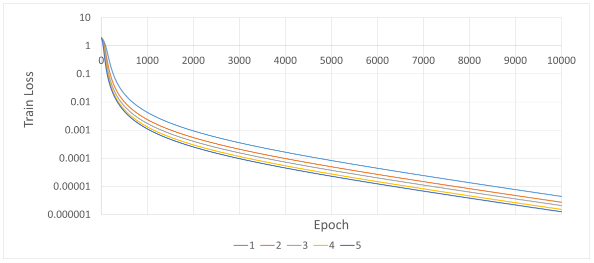
Figure 6.1: TAGCN: Train loss vs epoch for different number of layers

6.3.3.3 Nonlinearity

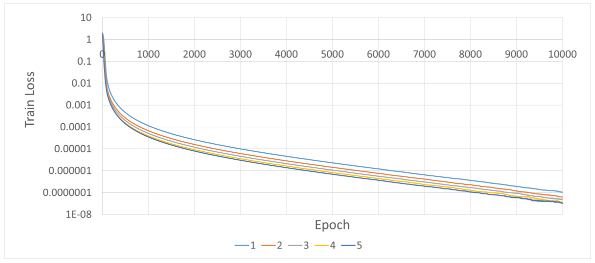
Fig. 6.3 shows TAGCN results with and without ReLU nonlinearity. In general, the results are similar, with training losses also converge to 0 with nonlinearity, albeit at a slower rate. This is due in part to difference in the default initialization.

6.4 Conclusion

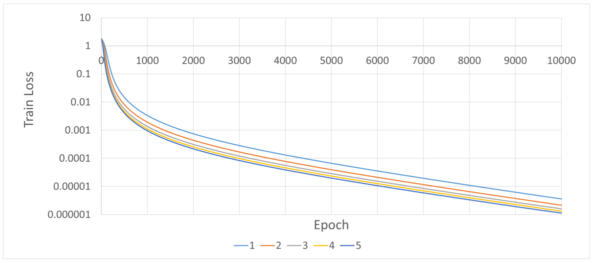
In conclusion, we show theoretically that training loss converges to a global minimum for linearized TAGCN. We first find the gradient of the loss with respect to the weights, then



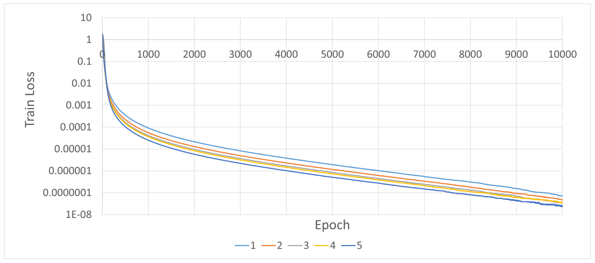
(a) CORA - 2 layers



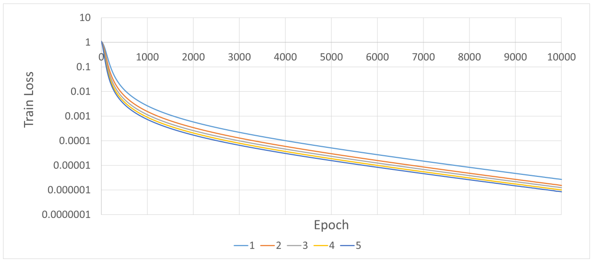
(b) CORA -4 layers



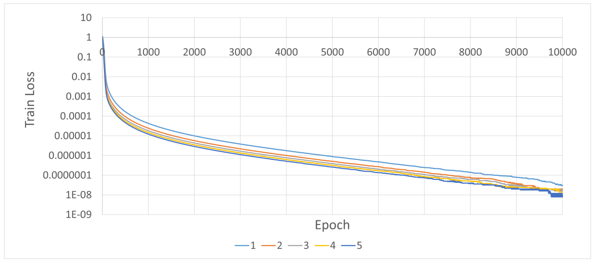
(c) CiteSeer - 2 layers



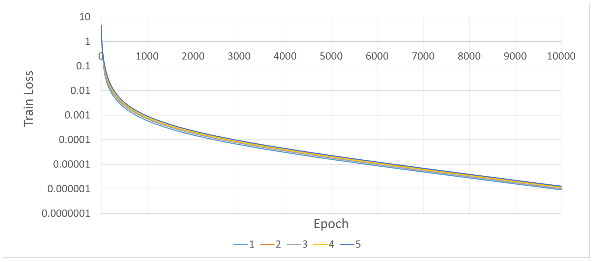
(d) CiteSeer -4 layers



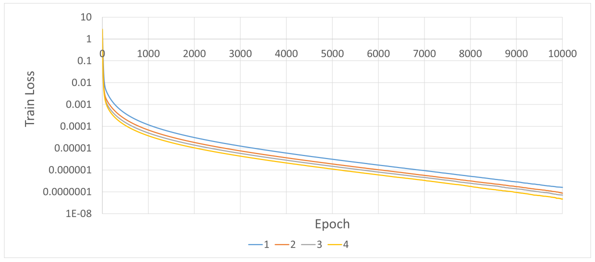
(e) PubMed - 2 layers



(f) PubMed -4 layers



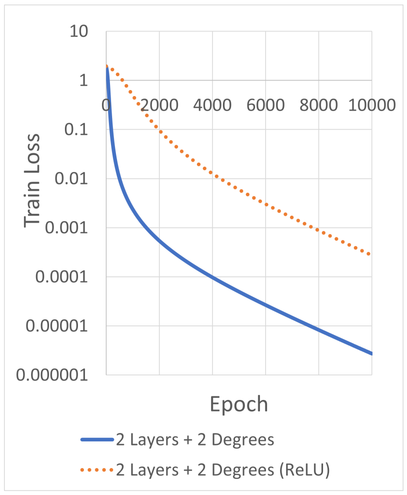
(g) Amazon Computer - 2 layers



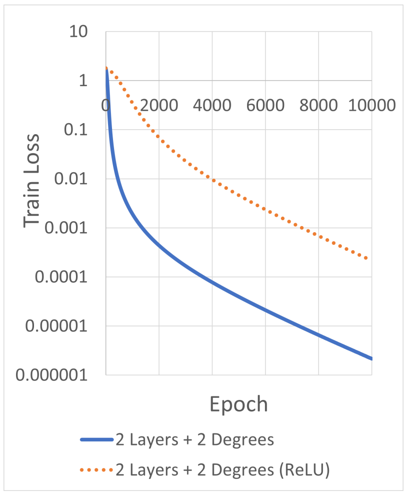
(h) Coauthor CS -2 layers

Figure 6.2: TAGCN: Train loss vs epoch for different degree of the polynomial filters

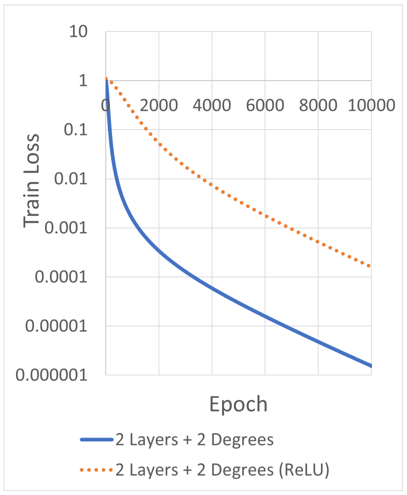
the dynamics induced with the weights and induced with the loss, finally we complete the proof assuming squared loss. Despite the non-convex objectives, training loss for 1-degree H-layer TAGCN, i.e., with degree 1 polynomial filter and H layers, is guaranteed to converge to global minimum at an exponential rate, faster with higher number of layers. With K-degree TAGCN, convergence is accelerated with higher degree K of the



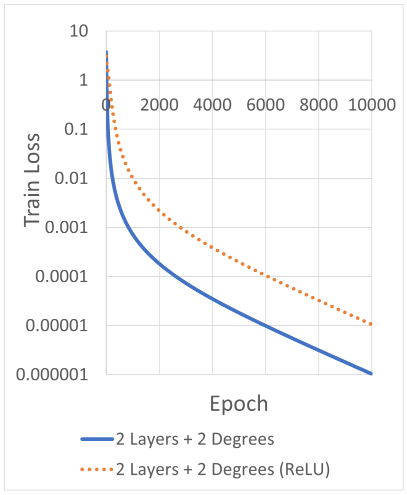
(a) CORA



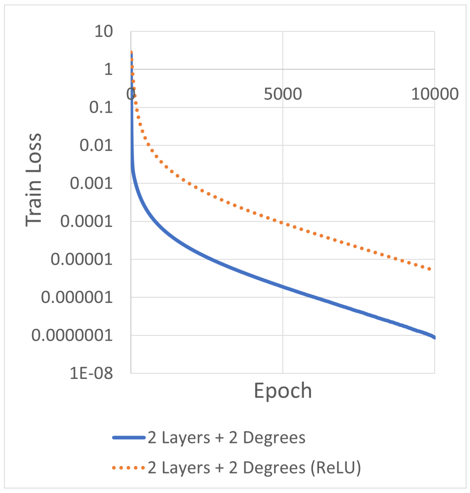
(b) CiteSeer



(c) PubMed



(d) Amazon Computer



(e) Coauthor CS

Figure 6.3: TAGCN with ReLU: Train loss vs epoch for different 2 layers + 2 degrees of polynomial filters

polynomial filters.

We experimentally validate our theory and show training convergence holds true in general for some real benchmark datasets. We plot the loss in log-linear graphs and show that increasing number of layers and degrees of the polynomial filters leads to decrease in loss. We also show that this training loss also converges with ReLU nonlinearity at a slower rate.

Chapter 7: COVID-19 Drug Discovery

The scientific community has united to combat the growing COVID-19 pandemic. According to [101], by 2020, there were more than 200 vaccines in development, 15 have begun large-scale (phase 3) clinical trials, and 2 have shown to be 95% effective (Pfizer and Moderna). Given the urgency of the situation, vaccine development and approval has been accelerated from the general timeline of 10-15 years [102] to under a year. Part of this acceleration is made possible due to the development of artificial intelligence (AI).

The first step of this process is drug discovery, whereby scientists typically spend 2-5 years to identify vaccine candidates [102]. Instead of testing most drugs *in vitro* in the lab, scientists are able to leverage computational methods to screen potential drugs *in silico*, saving significant time and physical resources, as there can be millions of compounds to test. In the last decade, scientists have found machine learning models to be a good predictor of potential drug candidates found experimentally. It has been shown that using deep learning, given enough data, can result in superior performance for antibiotic drug discovery [8]. For extensive reviews of AI applied to drug discovery, see [103, 104].

The scope of this chapter is one type of deep learning models, namely graph neural networks (GNNs), applied to COVID-19 drug discovery. This application is well studied, and we follow closely the approaches described in [8, 105]. In Section 7.1, we give a cursory summary of the drug-target interaction framework and the deep learning approach based on GNNs. Then we describe our specific approach combining topology adaptive graph neural networks (TAGCN [48]) and message-passing neural network in Section 7.3. We discuss five COVID-19 related datasets and experimental setup in Section 7.4. We present the results and analysis in Section 7.5. Finally, we conclude in Section 7.6. Content of this chapter is published in [106].

7.1 Background and Related Work

We divide the background into 3 parts: drug candidate discovery framework, drug-target interaction framework, and graph neural networks.

7.1.1 Drug Candidate Discovery Framework

Fig. 7.1 depicts the pipeline for drug candidate discovery using deep learning. According to [107], there is a large chemical space of at least 10^{63} drug-like molecules just by combining up to 30 carbon, nitrogen, oxygen, and sulfur atoms in different arrangements. Yet, the drugs that have made it into the clinic is a highly-selective fraction of that.

Many diseases are caused by protein binding or protein malfunction. For example, the virus of COVID, SARS-CoV-2 infects cells via spike protein binding to ACE2 protein surface receptors [108]. In general, these are what the drugs target. Pfizer-BioNTech and Moderna COVID-19 vaccines contain mRNA that give instructions to cells in our body on how to make a protein to trigger an immune response (which will prevent us from getting sick with COVID-19 in the future). Finding the right molecule is the job of the drug hunters / big pharmaceutical firms. To start, they need to find a "hit," i.e., a molecule that is effective before improving it through more experimentation and undergoing clinical trials.

The number of screening compounds is growing. Merck KGaA, one of the leading pharmaceutical firms, has Merck Accessible Inventory (MASSIV), a chemical space of 10^{20} molecules [109]. To navigate this ever-growing complexity, researchers have encoded the properties and applied deep learning to predict and design antibiotics and vaccines.

7.1.2 Drug-target Interaction Framework

One of the most important steps of drug discovery is predicting drug-target interactions (DTI). It characterizes the binding of chemical compounds to the protein targets. Among several, drug screening and drug repurposing are the two main tasks. Drug screening identifies ligand molecules that can bind to specific proteins, whereas drug repurposing finds new therapeutic uses of existing and available drugs. Traditionally, this is done

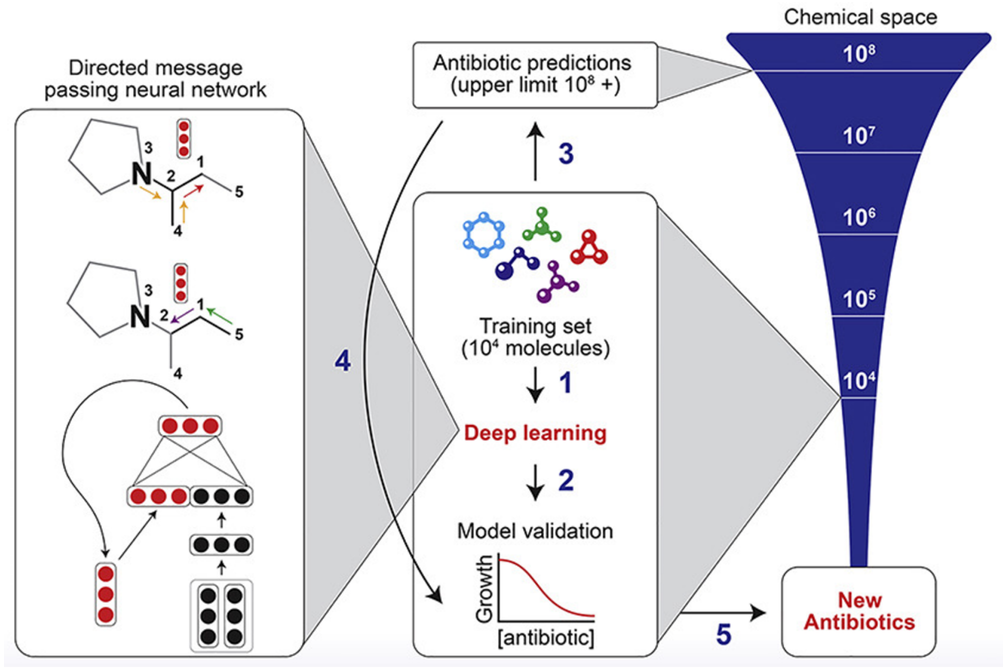

Directed message passing neural network
Antibiotic predictions (upper limit 10^8 +)
Chemical space
10^8
10^7
10^6
10^5
10^4
Training set (10^4 molecules)
1
Deep learning
2
Model validation
Growth
[antibiotic]
3
4
5
New Antibiotics

Figure 7.1: Machine learning in drug discovery. Image taken from [8].

experimentally in the lab. Nowadays, researchers generally use computational methods as a supplement, saving significant cost and time.

Deep learning has demonstrated superior performance than traditional computational methods [9]. In addition to classic chemical fingerprint drug encoders like Morgan and RDKit, there are many deep learning-based drug encoders that can be categorized into autoencoder, convolutional neural networks, recurrent neural networks, transformers, and GNNs. Fig. 7.2 shows the DTI framework from DeepPurpose [9]. On the left side, we have the encoder of the molecule, and, on the right side, we have the encoder for the protein receptor. The output of the encoder is feed into a decoder that produces a binding score or interaction probability.

7.1.3 Graph Neural Networks

Within deep learning, graph neural networks (GNNs) have shown promising results for drug discovery applications [110, 111]. They are motivated in part by convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which, as discussed in previous chapters, have yielded significant improvements in computer vision, natural language processing, and other Euclidean domains. Unlike CNNs and RNNs, GNNs are able to extract features via the graph structure or data topology, defined by its adjacency

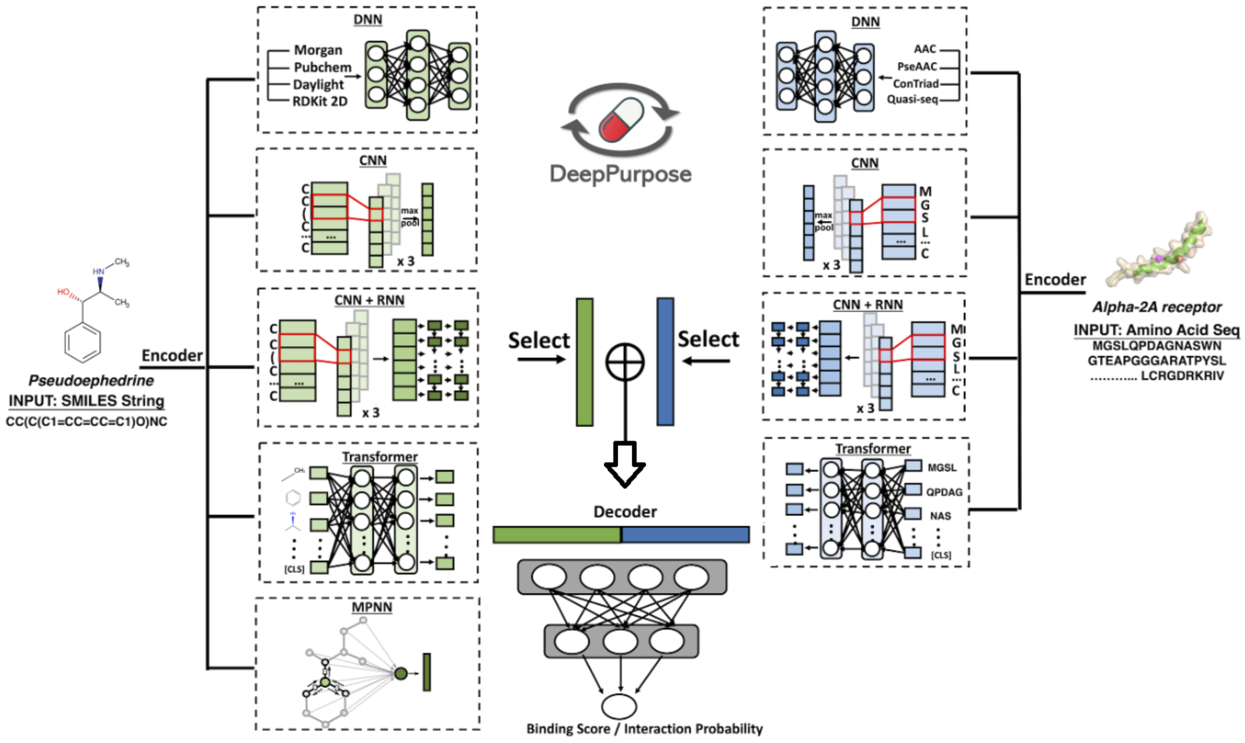


Figure 7.2: Drug-target interaction framework. Image from [9].

matrix. For a more in-depth review of GNNs, see Section 3.1 and [2].

7.2 Problem Definition

We cast the task of drug discovery as a graph classification task, just like it is defined in Section 5.1. The molecules can be represented as graphs $\mathcal{G}_s = (\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_N)$, and labels $\mathbf{Y}_s = (\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_N)$. We are given a subset of labels for training $\{\mathbf{Y}_i \mid i \in D_{\text{training}}\}$, and the task is to predict the rest $\{\mathbf{Y}_i \mid i \in D_{\text{testing}}\}$, where D_{training} and D_{testing} correspond to the train and test sets, respectively.

Each graph $\mathcal{G}_i$ consists of a set of atoms/nodes V_i with $|V_i| = N_i$, signal $\mathbf{X}_i \in \mathbb{R}^{N_i \times C_0}$ on the nodes (where C_0 is the number of features on each node), a set of atomic bonds/edges $\mathcal{E}_i$, and an adjacency matrix $\mathbf{A}_i$. $[\mathbf{A}_i]_{jk} = 0$ unless there is an bond/edge $e = (j, k)$ connecting node j and node k.

7.3 Methods

In this section, we describe our approach combining two GNN convolution methods for drug discovery: topology adaptive graph neural networks and message-passing neural network. We also describe our methods dealing with class imbalances, extra features, and different losses.

7.3.1 Topology Adaptive Graph Convolutional Neural Networks

We focus on the convolutional layer of the GNN architecture. The topology adaptive graph convolution network (TAGCN) implementation of graph convolution [48] uses the polynomial filter coefficients as learnable weights. We recall from (3.2), the general form of the TAGCN graph convolutional layer is:

$$\mathbf{X}^{(\ell+1)} = \sigma\left(\sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}^{(\ell)} \mathbf{W}_k^{(\ell)}\right). \tag{3.2}$$

where K is the degree of the graph polynomial filter and a hyperparameter of the model. TAGCN allows learning of more complex functions with deeper models. See Fig. 3.2 for an example of graph convolutional filter of degree 2.

7.3.2 Message Passing Neural Networks

Message passing neural networks (MPNNs) are a generalization of GNNs, first formulated by [112]. Following the same definition given above, we define $\boldsymbol{h}_i^l \in \mathbb{R}^C$ to be the state of node i at t-th time step and $\boldsymbol{h}_i^0 \in \mathbb{R}^C$ to be the initialized state (from splitting $\mathbf{X}^{(0)}$ by nodes).

One GNN layer can be expressed using message passing neural network:

$$\boldsymbol{m}_{i\to j}^{\ell+1} = f_\alpha^\ell(\boldsymbol{h}_i^\ell, \boldsymbol{h}_j^\ell, \mathbf{A}_{ij}) \qquad \text{(propagation)} \tag{7.1}$$

$$\boldsymbol{h}_j^{\ell+1} = f_\beta^\ell(\{\boldsymbol{m}_{i\to j}^{\ell+1} | i \in \mathcal{N}_j\}, \boldsymbol{h}_j^\ell) \qquad \text{(aggregation)} \tag{7.2}$$

where ℓ is the time step (or layer number in GNN); f_α^ℓ and f_β^ℓ are parametrized functions like neural networks; $\boldsymbol{m}_{i\to j}^{\ell+1}$ is the propagated information from node i to node j. For example, f_α^ℓ can be a MLP followed by a nonlinear activation, and $\boldsymbol{m}_i^{\ell+1}$ is the aggregated information with $\mathcal{N}_i$ representing all the incoming neighbors of node i:

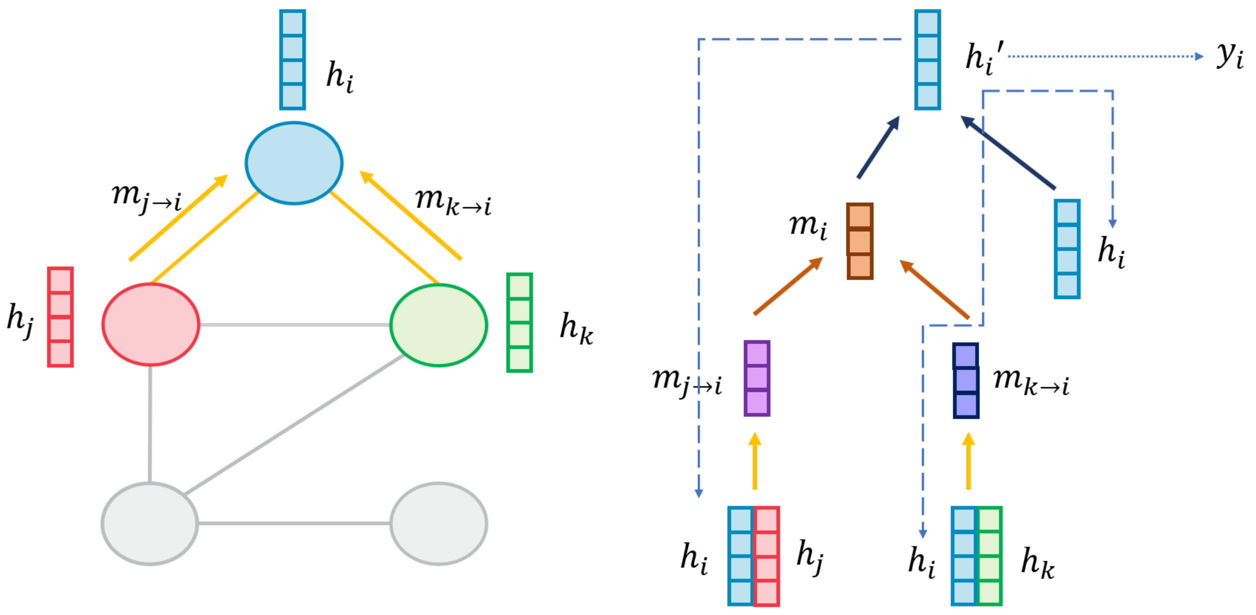


Figure 7.3: Propagation and aggregation steps.

$$\boldsymbol{m}_i^{\ell+1} = \sum_{j\in N_i} \boldsymbol{m}_{j\to i}^{\ell+1} \tag{7.3}$$

To update the hidden state at $\ell+1$ for each node, we can also use a MLP or a recurrent neural network like a GRU or LSTM for f_β:

$$\boldsymbol{h}_i^{\ell+1} = f_\beta(\boldsymbol{m}_i^{\ell+1}, \boldsymbol{h}_i^{\ell}) \tag{7.4}$$

After $\mathcal{L}$ time steps/layers, we get the final states $\boldsymbol{h}_i^{\mathcal{L}}$ and predict the graph label using a readout function R (which can also be MLPs):

$$\hat{y} = R(\{\boldsymbol{h}_i^{\mathcal{L}} \mid \forall i \in \mathcal{V}\}) \tag{7.5}$$

Then apply a loss function like cross entropy loss $\mathrm{CE}(\hat{y}, y)$. See fig. 7.3 for a visualization of this process.

This method is essentially the same as convolution in GNN. By breaking it down into propagation and aggregation, additional functions and parameters can be easily written out at each step. For example, in our study, we incorporate graph-level RDKit features into the atoms, and consider different edge types.

7.3.3 Topological adaptive message passing neural networks

Our approach is applying TAGCN to MPNN to get topology adaptive message passing neural network. Instead of summing just the direct neighbors in (7.3), we aggregate nodes up to a degree K hops away (including 0 hop) and apply the nonlinear activation after the summation. For example, for $K = 2$, we have

$$\boldsymbol{m}_i^{\ell+1} = \sum_{j \in N_i} (w_2 \boldsymbol{m}_{j \to i}^{\ell+1} + w_1 \sum_{r \in N_j} \boldsymbol{m}_{r \to j}^{\ell+1}) + w_0 \boldsymbol{m}_i^{\ell} \tag{7.6}$$

MPNN generalization enables us to consider more variants such as passing messages on the nodes.

7.3.4 Class Imbalance Solutions

For drug discovery, there is severe class imbalance, i.e., the number of hits is generally a small proportion of the number of no hits. This is expected especially for novel proteins/virus. As a result, we consider several solutions.

First, it is common practice to use a different performance metric called the area under the receiver operating characteristic curve (ROC-AUC). ROC is a plot of false positives (1- specificity) vs true positives (sensitivity), plotted using different decision thresholds on the score produced by the classifier. An example is shown in Fig. 7.4. AUC is the area under the curve for false positive rate between 0 and 1 (which is 0.897 here).

For the loss functions, we consider several alternatives. First, we consider weighted loss function, modified from (5.6) for the binary classification (y_i is either 0 or 1):

$$L(\widehat{y}_i, , y_i) = w_1 y_i \log(S(\widehat{y}_i)) + w_0(1 - y_i)\log(1 - S(\widehat{y}_i)) \tag{7.7}$$

where $S(\cdot)$ is the softmax function, w_0 and w_1 are the assigned weights of each class. We start with $w_0 = 1$, and $w_1 = \frac{N_0}{N_1}$ where N_0 and N_1 are the number of samples in class 0 and class 1, respectively.

We also experiment with another loss function called Matthews correlation coefficient [113], which [114] has shown to be more reliable in scenarios with class imbalance. Besides false positives (FP) and true positives (TP), it also considers true negatives (TN) and false negatives (FN). The metric is as follows:

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN}}} \tag{7.8}$$

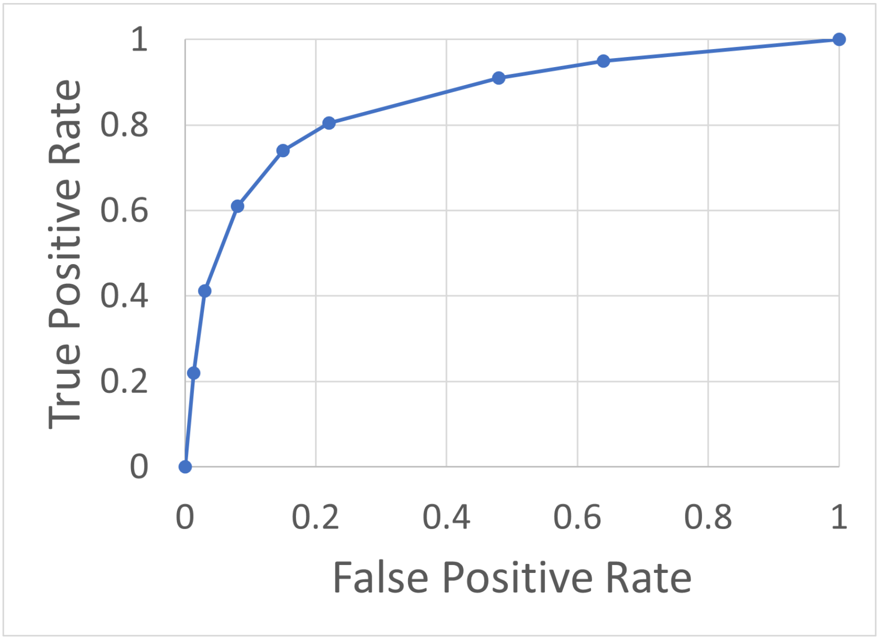


Figure 7.4: An example of receiver operating characteristic (ROC) curve.

For the actual loss function, we multiply each term by its probability, e.g., true positives and false negatives are $\widehat{y}_i \times y_i$ and $(1 - \widehat{y}_i) \times (1 - y_i)$, respectively. A small ϵ is added to each term for numeric stability.

7.4 Experiments

In this section, we describe the datasets and the experimental setup.

7.4.1 Datasets

As shown in table 7.1, we consider five COVID-19 related datasets, with total indicating the number of molecules and hits indicating whether it is effective in some way (e.g., inhibiting of SARS-CoV 3CL protease, which is the main enzyme found in the coronavirus).

Mpro Xchem [115] is a list of chemical fragments or blocks of chemical structures taken from 5 fragment libraries. The fragments are screened for 3CL protease binding using XChem crystallography.

Amu SARS dataset [116] is a list of FDA-approved compounds screened against SARS-CoV-2 *in vitro*. The hits represent potential inhibitors of SARS-CoV-2 replication.

Ellinger [117] is a list of compounds screened against SARS-CoV-2 *in vitro* using a large scale drug repurposing collection, including 3488 that had undergone clinical investigations (e.g., phases 1-3).

Broad SARS [118] is a list of compounds screened against SARS-CoV 3CL protease experimentally from Drug Repurposing Hub at Broad Institute. SARS-CoV is also a coronavirus with similar symptoms. It led to a pandemic in 2003 that was more severe in terms of mortality rate, but less transmissible.

AID1706 SARS [119] is a much longer list of compounds screened against SARS-CoV *in-vitro* via fluorescence, containing 41 hits from Broad SARS.

The features on the atoms include atomic number, number of bonds, chirality, formal charge, number of hydrogen atoms, hybridization (sp, sp2, etc.), aromaticity, and atomic mass. The features on the edges are bond type (single, double,etc.), whether bond is conjugated, whether bond is part of a chemical ring, and stereo. All features are one-hot encodings except for atomic mass.

Table 7.1: COVID-19 Related Datasets

Name	Total	Hits
Mpro Xchem	880	78
Amu SARS	1,484	88
Ellinger	5,632	67
Broad SARS	5,671	41
AID1706 SARS	290,767	446

7.4.2 Experimental Setup

In our experiments, we use the state-of-the-art Directed-MPNN (D-MPNN) model from [105] as baseline. Like TAGCN, directed means the adjacency matrix is not symmetric. We use the same initial featurization as D-MPNN. We perform 5-fold cross-validation using scaffold split and evaluate our results using common benchmark area under the receiver operating characteristic curve (ROC-AUC). We compare results with and without the recommended graph-level features RDKit. To address class imbalance, we apply weighted cross-entropy loss function and Matthews correlation coefficient (see Section 7.3.4). We consider several hyperparameters such as number of layers (convolution and linear), number of channels, degree of the graph polynomial filter, and dropout rates. We perform hyperparameter optimization via Bayesian Optimization using the Hyperopt package [120].

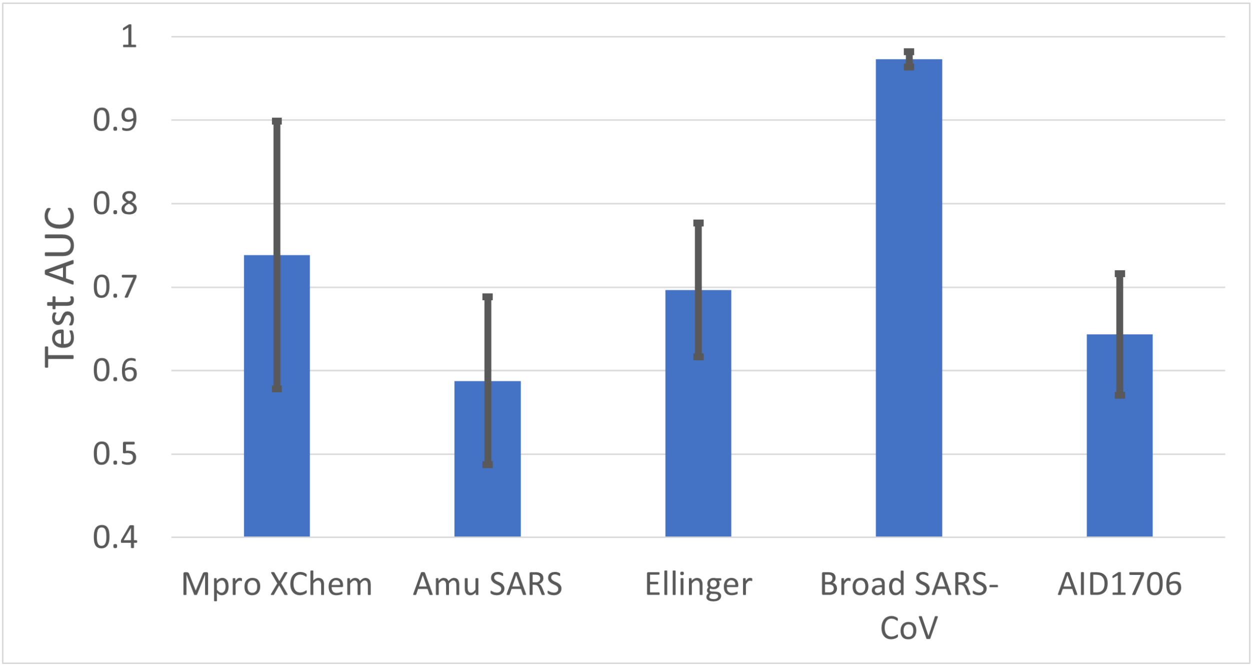


Figure 7.5: Simple classifier on most common atoms for different datasets. Blue bars represent average test AUCs. Vertical black lines on the bars represent standard deviations.

We also consider one simple metric, by classifying using different number of the 7 most common atoms. For each graph, we create a feature vector consisting of the number of carbon, hydrogen, oxygen, nitrogen, sulfur, chloride, and fluoride atoms. Then we apply logistic regression, linear support vector machine, and SGD Classifier (for AID1706, as there are more than 100,000 samples), all from scikit-learn library [87].

7.5 Results

Fig. 7.5 shows the results on the simple classifier on the most common atoms. There is a spectrum of results. For Broad SARS, we can achieve 97% AUC, and for all others AUC around or below 75%.

Fig. 7.6 shows the optimized results of D-MPNN+TAGCN vs state-of-the-art D - MPNN for five COVID-related datasets, quantified by test AUC. In general, we observe that D-MPNN+TAGCN yields better performance than D-MPNN for all datasets, regardless of whether extra feature (RDKit) is applied. RDKit improves the performance for all except the Amu SARS dataset, and leads to higher variance for all except the Mpro XChem dataset (since extra features were used).

A comparison between Fig. 7.6 and Fig. 7.5 suggests that if the classifier on the metric (number of common atoms) performs well, D-MPNN+TAGCN and D-MPNN both perform well with/without RDKit. For example, for Broad SARS, the metric achieves 0.97 AUC while GNN approaches achieve near 1 AUC.

Fig. 7.7 shows the test AUC for D-MPNN+TAGCN with/without sumpool. It leads to slight improvement for the MPro XChem dataset only.

We look into two hyperparameters of graph neural network architectures: number of layers and degree of the polynomial filters.

Fig. 7.8 shows the test AUC of D-MPNN vs number of layers. In general, optimal AUC occurs within the first 5 layers. There is no significant change as the number of layers increases for all datasets, suggesting that the D-MPNN do not over-smooth on the dataesets.

Fig. 7.9 shows the test AUC of D-MPNN+TAGCN vs number of layers and the degree of polynomial filters. In general, for the initial degrees, the AUC with respect to the number of layers is similar to that of D-MPNN. As the degree of polynomial increases, at higher number of layers, over-smoothing occurs. The initial test AUC shapes of Ellinger and Mpro XChem are more similar than Amu Sars. Their test AUCs do not start dropping till around polynomial filters of degree 6 and 5 layers, while for Mpro XChem, the relative drop in AUC is much sooner at around polynomial of degree 4 and 4 layers. For Broad SARS, test AUC is near 1 till after 9 layers and polynomial of degree 9

7.6 Conclusions

In this chapter, we have introduced a novel approach for molecular property prediction by combining two existing GNN methods. We performed experiments comparing this approach to the state-of-the-art D-MPNN baseline. Our model (D-MPNN+TAGCN) consistently outperforms this baseline on five coronavirus datasets, implying that these methods may aid in COVID-19 drug discovery.

We also performed an analysis of the architectures. Using just D-MPNN, there is no significant difference with respect to just changing the number of the layers. With D-MPNN+TAGCN, as the degree of the polynomial filters increases, test AUC decreases with Amu SARS having a relatively steeper decline, suggesting that these molecules have smallworld characteristics with a difference in edge rewiring probability between classes.

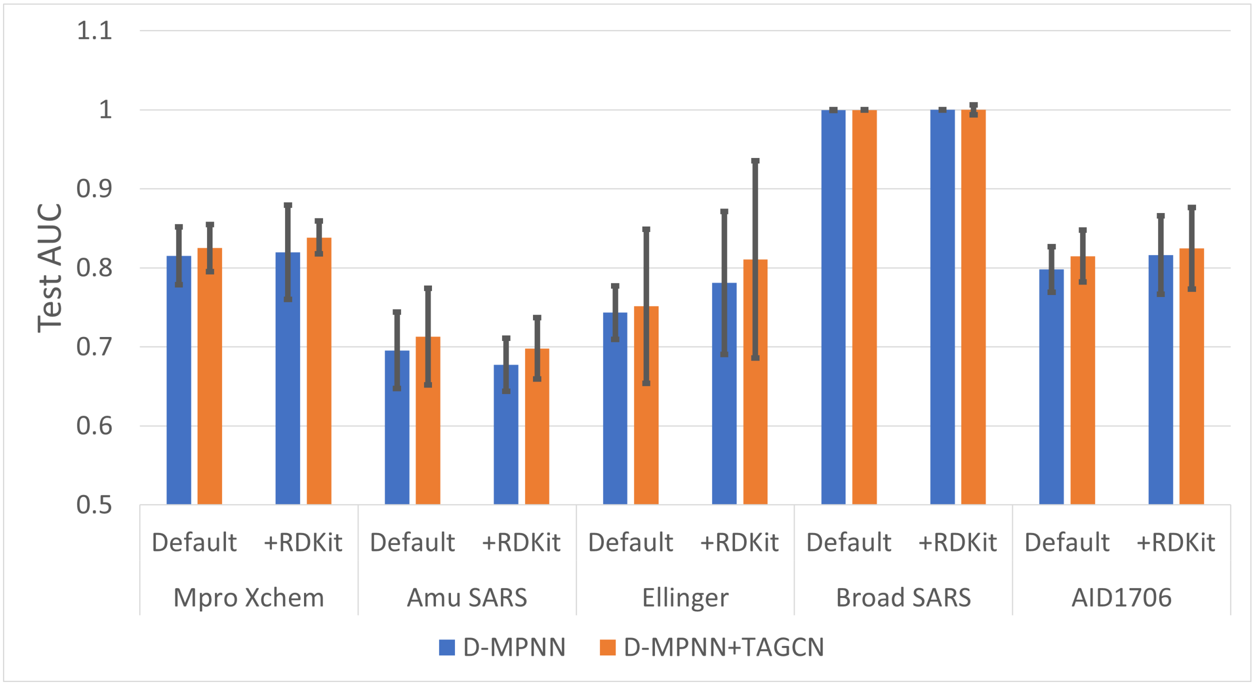


Figure 7.6: D-MPNN+TAGCN vs D-MPNN for COVID related datasets. Blue and orange bars represent average test AUCs. Vertical black lines on the bars represent standard deviations.

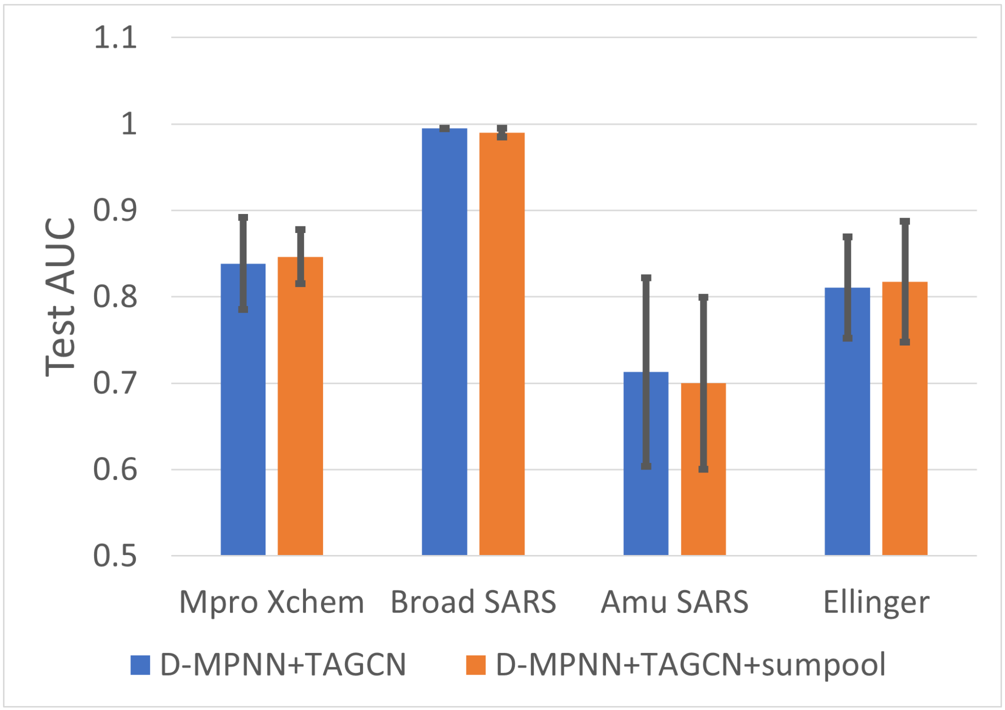


Figure 7.7: D-MPNN+TAGCN vs D-MPNN for COVID related datasets with sumpooling. Blue and orange bars represent average test AUCs. Vertical black lines on the bars represent standard deviations.

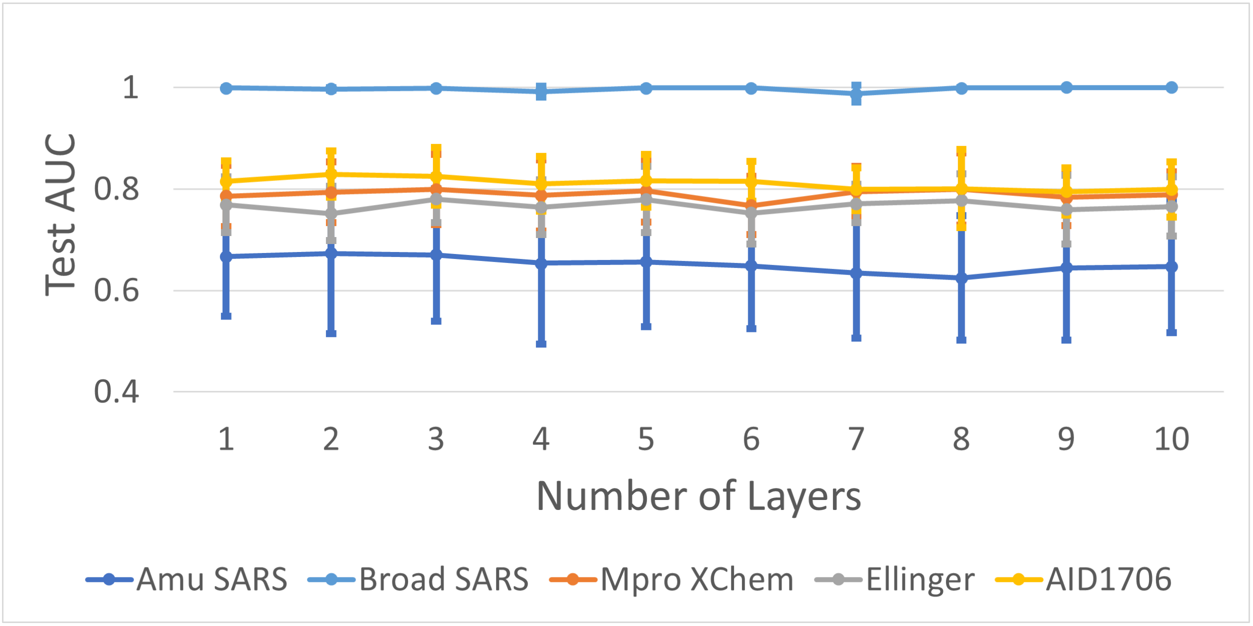
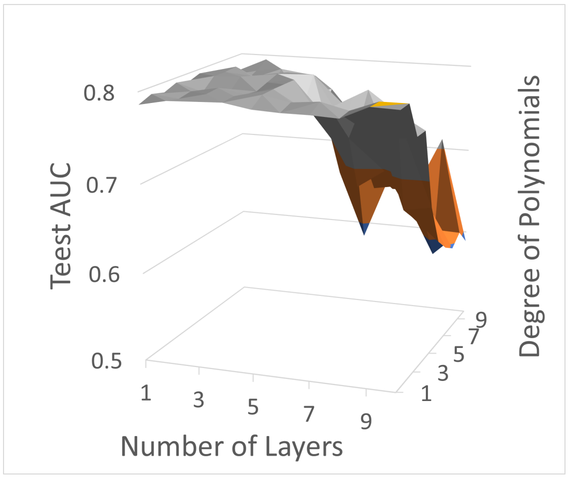
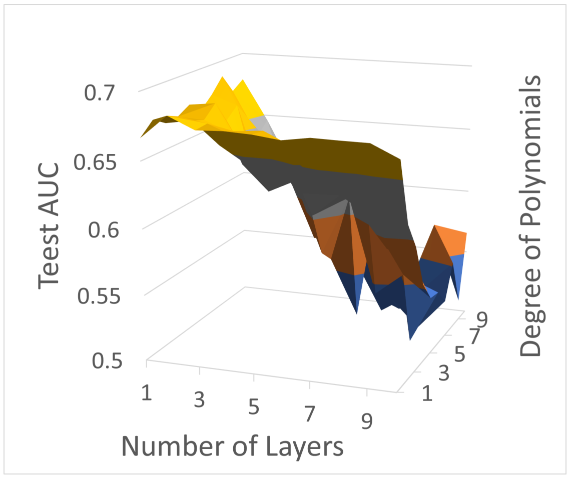


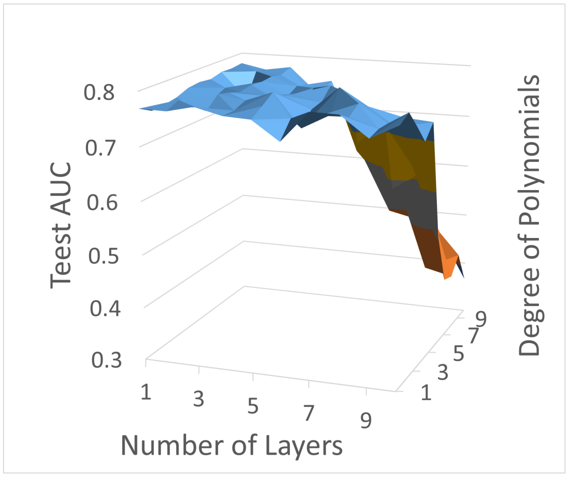
Figure 7.8: D-MPNN: Test AUC vs number of layers for different COVID-related datasets. Colored horizontal and vertical lines represent average test AUCs and standard deviations, respectively.



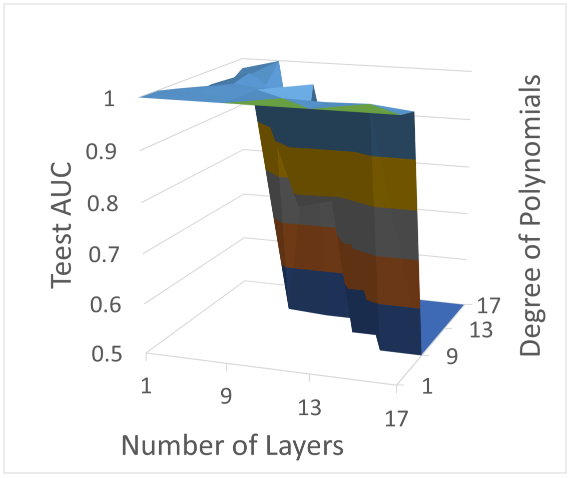
(a) Mpro XChem



(b) Amu SARS



(c) Ellinger



(d) Broad SARS

Figure 7.9: D-MPNN+TAGCN: Test AUC vs number of layers and degree of the polynomial filters for different COVID-related datasets

Chapter 8: Conclusions

This thesis aims to be a step towards understanding geometric deep learning, the application of deep learning to non-Euclidean domains. Towards this end, we have shown that the data graph structure has an important effect on the performance of graph neural networks for node and graph classification tasks. We believe that by considering the graph structure, researchers can design more suitable architectures with better performance.

We start with the formalization of a geometric deep learning algorithm called graph neural networks (GNNs) in Chapter 3. We organize its architecture into convolutional layer, aggregation layer, and pooling layer. The chapter relates the graph convolutional layer to graph signal processing, and considers two flavors of approaches: spectral approach typified by graph convolutional networks (GCNs) and spatial approach typified by topology adaptive graph convolutional networks (TAGCNs). In general, TAGCN requires fewer number of layers than GCN, with moderate degrees of the polynomial filters.

In Chapter 4, we apply GNNs to the task of node classification. In general, for the two main GNN architectures that we studied (GCN and TAGCN), not many layers are needed to achieve optimal performance (compared with computer vision and natural language processing). TAGCN requires fewer number of layers than GCN, with low degree of the polynomial filters. Unlike graph classification (studied in Chapter 5), graph signal is necessary and important. For some real datasets, classifying using a simple estimator on the graph signals can outperform GNNs. For synthetic datasets, Erdős-Rényi and preferential attachment models have similar test accuracy curves for both GCN and TAGCN with respect to the number of layers and the degree of the polynomial filters. For smallworld model, TAGCN's filters play an important role in achieving the optimal accuracy and accelerating the effect of over-smoothing.

For real datasets, accuracy is significantly improved when the graph is taken into consideration. We relate simple metrics on real datasets to the performance of these models. We show, for datasets with higher difference in % of intraclass and interclass edges, the effect of over-smoothing due to number of layers is smaller. We also show that over-smoothing occurs slower vs the degree of polynomial filters for datasets with low average degree. For synthetic datasets, we relate the performance to different graph characteristics. We find that, in general, high intraclass edge ratio not only leads GNN achieving higher accuracy, but also leads to them resisting over-smoothing more for both number of layers and degree of polynomial filters (which has higher optimal values for lower layers, and lower optimal values for higher layers). Changing the graph signals by increasing the interclass difference between the means of the graph signal improves the accuracy while not affecting the optimal number of layers and degree of polynomial filters. Increasing the number of nodes for the preferential attachment model only lowers the variance of the test accuracy for both architectures if all other graph characteristics remain the same.

In Chapter 5, we apply GNNs to the task of graph classification. Unlike with node classification, graph structure plays a more important role than the graph signal. GNNs can often classify graphs using just the graph structures of the different classes if they are distinct enough. For real datasets, we relate simple network metrics and signal statistics to the performance of these models. We show for biological network datasets, classifiers on number of edges or number of nodes can lead to better or similar performance as graph neural networks. For social network datasets, signal statistics can perform well. Based on these observations, we were able to apply simple modifications to GCN and TAGCN to improve their performance (sumpool and degree-aware TAGCN).

For real datasets, we also look into the impact of the GNN architecture on its performance. In general, datasets with high average degree and low average diameter require less number of layers to achieve optimal performance. Over-smoothing also occurs for graph classification. It happens at lower number of layers if the graphs have high average diameter though there are some exceptions, depending on the graph signal. TAGCN tends to perform better with higher degrees of the polynomial filters at low number of layers on datasets with low average degree.

For synthetic datasets, Erdős-Rényi and preferential attachment models again have similar test accuracy curves for both GCN and TAGCN. For smallworld model, more than 1 layer is needed to achieve good performance if the edge rewiring probability is distinct for different classes. We also find that simple network metrics that capture different interclass graph characteristics can perform better than GNN (e.g., clustering coefficient performs better than TAGCN for capturing the smallworld's edge rewiring probability). This suggests that GNNs are not optimal for capturing Erdős-Rényi's edge creation probability.

Just like real datasets, we also show that simple modification (degree-aware TAGCN) can greatly improve the performance in capturing interclass difference in degree. However, another popular variant (graph attention network) does not perform better than TAGCN on capturing these graph characteristics. Finally, we show that while TAGCN can perform well when graph signal and graph structure have an AND relationship, it fails when they have an XOR relationship.

We then study the training convergence for node classification in Chapter 6. Following previous works on gradient dynamics [96, 97, 98], we show theoretically that training loss converges to a global minimum for linearized TAGCN. We first find the gradient of the loss with respect to the weights, then the dynamics induced with the weights and induced with the loss, finally we complete the proof assuming squared loss. Despite the non-convex objectives, training loss for 1-Degree H-Layer TAGCN, i.e., with degree 1 polynomial filter and H layers, is guaranteed to converge to global minimum at an exponential rate, faster with higher number of layers. With K-Degree TAGCN, convergence is accelerated with higher degree K of the polynomial filters.

We experimentally validate our theory and show training convergence holds true in general for some real benchmark datasets. We plot the loss in log-linear graphs and show that increasing the number of layers and degrees of the polynomial filters leads to decrease in loss. We also show that this training loss also converges with ReLU nonlinearity at a slower rate.

Finally, we apply the architecture of TAGCN to a COVID-19 case study in Chapter 7. We introduce a novel approach for molecular property prediction by combining two existing GNN methods. Our model (D-MPNN+TAGCN) consistently outperforms the D-MPNN state-of-the-art baseline on five coronavirus datasets. This is likely because with TAGCN the model not only passes residual connections from previous layers but also considers different neighborhood ranges for each nonlinearity for better structure-aware representation. We also perform an analysis of the architectures. Using just D-MPNN, there is no significant difference with respect to just changing the number of layers. With D-MPNN+TAGCN, as the degree of the polynomial filters increases, test AUC decreases with Amu SARS having a relatively steeper decline, suggesting that these molecules have smallworld characteristics with a difference in edge rewiring probability between classes.

In the course of this thesis, we published [2, 45, 51, 75, 78, 106] that report our results.

8.1 Future Directions

It is still an open question how the data graph structure affects GNN performance. In this thesis, we only consider two of the many possible graph-related tasks with three main GNN architectures. There are several interesting future directions and we group them into five categories: GNN architecture, graph-related tasks, explanation methods, GSP connection, and training convergence.

8.1.1 GNN Architecture

This thesis applies three graph convolutional models that represent three common types of graph convolution models: GCN typifies spectral-based models, TAGCN typifies spatial-based models, and graph attention network typifies attention-based models. Within the spatial-based model, we consider a more general propagation based model (message passing neural network) in Chapter 7, and a spectral domain convolution in Chapter 3.

There are many other convolution models that can be applied. For example, graph isomorphic network (GIN) [24] passes the residual connection to 1 plus a trainable parameter ϵ in addition to neighborhood aggregation. This theoretically guarantees to be most expressive among the class of GNNs. However, [121] reports GIN performing worse than both GCN and GAT for the citation datasets (CORA, CiteSeer, PubMed). Another interesting model is Gaussian mixture model (GMM), which normalizes the neighborhood aggregation with a learnable Gaussian kernel (the mean vector and covariance matrix are trainable parameters). It would be interesting to see if GIN and GMM have similar behavior as GCN in regard to the number of layers, and if they can capture certain graph statistics with/without modifications.

Besides the convolutional layer, there are many variants of pooling and aggregation layers. Reference [51] shows that graph pooling improves graph classification accuracy on some benchmark datasets. We find that graph pooling performs better when combined with TAGCN vs GCN. Reference [45] links pooling to sampling methods in graph signal processing for both the vertex domain and the spectral domain. Pooling makes it possible for the model to learn hierarchically and to accelerate the training process.

8.1.2 Graph-related Tasks

Node classification and graph classification are just two of the many graph-related tasks that GNNs/machine learning can be applied for. Some other tasks of interest include link prediction, graph embedding, graph sampling/generation, community detection, outlier detection, graph matching, graph clustering, point cloud classification, and graph question answering. GNNs have also been applied to natural language processing and computer vision. At the base of all these methods is graph convolution, typified by GCN and TAGCN. To solve these tasks, researchers have developed many geometric deep learning architectures that are extensions of these, such as graph autoencoders, recurrent graph neural networks, and spatial-temporal graph neural networks (e.g., [122]). For future work, it would be interesting to study how the graph structures play a role for these tasks.

8.1.3 Explanation Methods

We have discussed several interpretability methods for GNNs in Section 1.1.6 of the introduction of this thesis. They are used mostly for subgraph detection task, whereas here we are trying to detect the global characteristics in graph classification and individual node classes for node classification. One future direction is to apply these methods for the proposed methods and the synthetic graphs.

8.1.4 GSP Connection

In Chapter 2, we show how graph signal processing (GSP) can be extended to convolutional neural networks components to get graph convolutional neural networks based on our work in [2]. GSP can also be used to tackle research directions for GNNs, including model depth, over-smoothing [123], scalability, heterogenity, dynamicity, and interpretability (see [21, 22] for more details).

8.1.5 Training Convergence

In Chapter 6, we prove the training convergence for TAGCN for node classification and validate it experimentally on real datasets. We outline a sketch of the proof for graph classification in Appendix A. Directions for future work include completing this proof, and proving this with non-linear models and other loss functions.

Bibliography

[1] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, "Benchmark data sets for graph kernels," 2016, http://graphkernels.cs.tu-dortmund.de. viii, xi, 37, 61, 66, 68

[2] M. Cheung, J. Shi, O. Wright, L. Y. Jiang, X. Liu, and J. M. F. Moura, "Graph signal processing and deep learning: Convolution, pooling, and topology," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 139–149, 2020. ix, x, 21, 26, 27, 28, 129, 143, 144

[3] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *32nd International Conference on Neural Information Processing Systems (NeuRIPS)*. Curran Associates Inc., 2018, pp. 4805–4815. [Online]. Available: http://dl.acm.org/citation.cfm?id=3327345.3327389 x, 27

[4] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 3734–3743. x, 27

[5] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *32nd AAAI Conference on Artificial Intelligence*, 2018. [Online]. Available: https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17146 x, 27

[6] H. Gao and S. Ji, "Graph U-Nets," in *36th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, Cali-

fornia, USA: PMLR, 09–15 Jun 2019, pp. 2083–2092. [Online]. Available: http://proceedings.mlr.press/v97/gao19a.html x, 27

[7] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," pp. 361–362, 2009, 3rd International AAAI Conference on Weblogs and Social Media. x, 31

[8] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann, V. M. Tran, A. Chiappino-Pepe, A. H. Badran, I. W. Andrews, E. J. Chory, G. M. Church, E. D. Brown, T. S. Jaakkola, R. Barzilay, and J. J. Collins, "A deep learning approach to antibiotic discovery," *Cell*, vol. 180, no. 4, pp. 688 – 702.e13, 2020. xiii, 126, 128

[9] K. Huang, T. Fu, L. Glass, M. Zitnik, C. Xiao, and J. Sun, "Deeppurpose: a deep learning library for drug-target interaction prediction and applications to repurposing and screening," *Computing Research Repository*, vol. abs/2004.08919, 2020. xiii, 128, 129

[10] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997. 2

[11] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006. 2

[12] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org. 3

[13] Y. Wang, B. Widrow, L. A. Zadeh, N. Howard, S. Wood, V. C. Bhavsar, G. Budin, C. Chan, R. A. Fiorini, M. L. Gavrilova *et al.*, "Cognitive intelligence: Deep learning, thinking, and reasoning by brain-inspired systems," *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, vol. 10, no. 4, pp. 1–20, 2016. 3

[14] B. A. Richards, T. P. Lillicrap, P. Beaudoin, Y. Bengio, R. Bogacz, A. Christensen, C. Clopath, R. P. Costa, A. de Berker, S. Ganguli, C. J. Gillon, D. Hafner, A. Kepecs, N. Kriegeskorte, P. Latham, G. W. Lindsay, K. D. Miller, R. Naud, C. C. Pack, P. Poirazi, P. Roelfsema, J. Sacramento, A. Saxe, B. Scellier, A. C. Schapiro, W. Senn, G. Wayne, D. Yamins, F. Zenke, J. Zylberberg, D. Therien, and K. P. Kording, "A deep learning framework for neuroscience," *Nature Neuroscience*, vol. 22, no. 11, pp. 1761–1770, Nov 2019. [Online]. Available: https://doi.org/10.1038/s41593-019-0520-2 3

[15] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. 3

[16] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: Going beyond Euclidean data,” *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, July 2017. 3, 4

[17] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *5th International Conference on Learning Representations, (ICLR) 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl 4, 22, 23, 24, 34, 36, 65

[18] A. Sandryhaila and J. M. F. Moura, “Discrete signal processing on graphs,” *IEEE Trans. Signal Processing*, vol. 61, no. 7, pp. 1644–1656, Apr. 2013. 5, 16, 17

[19] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains,” *IEEE Signal Processing Magazine*, vol. 30, pp. 83–98, May 2013. 5

[20] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, “Graph signal processing: overview, challenges, and applications,” *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, May 2018. 5, 13, 16, 17, 18

[21] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, “Graph neural networks: A review of methods and applications,” *CoRR*, vol. abs/1812.08434, 2018. [Online]. Available: http://arxiv.org/abs/1812.08434 5, 30, 61, 144

[22] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” *CoRR*, vol. abs/1901.00596, 2019. [Online]. Available: http://arxiv.org/abs/1901.00596 5, 30, 61, 144

[23] S. Verma and Z.-L. Zhang, “Stability and generalization of graph convolutional neural networks,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1539–1548. [Online]. Available: https://doi.org/10.1145/3292500.3330956 5

[24] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, (ICLR) 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. 5, 66, 143

[25] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and Leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, Jul. 2019, pp. 4602–4609. 5, 66

[26] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 2156–2167. 5, 66

[27] L. Ruiz, L. Chamon, and A. Ribeiro, "Graphon neural networks and the transferability of graph neural networks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1702–1712. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/12bcd658ef0a540cabc36cdf2b1046fd-Paper.pdf 5

[28] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=r1Ue8Hcxg 5

[29] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33014780 5

[30] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, "Online learning rate adaptation with hypergradient descent," in *Sixth International Conference on Learning Representations (ICLR), Vancouver, Canada, April 30 – May 3, 2018*, 2018. 5

[31] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial intelligence review*, vol. 18, no. 2, pp. 77–95, 2002. 5

[32] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019. 5

[33] M. McGough, “How bad is sacramento’s air, exactly? google results appear at odds with reality, some say.” Aug 2018. [Online]. Available: https://www.sacbee.com/news/california/fires/article216227775.html 6

[34] R. Wexler, “When a computer program keeps you in jail,” Jun 2017. [Online]. Available: https://www.nytimes.com/2017/06/13/opinion/how-computers-are-harming-criminal-justice.html 6

[35] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann, “Explainability methods for graph convolutional neural networks,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 764–10 773. 6

[36] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, “Gnnexplainer: Generating explanations for graph neural networks,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 9240–9251. 6

[37] T. Schnake, O. Eberle, J. Lederer, S. Nakajima, K. T. Schutt, K.-R. Mueller, and G. Montavon, “Higher-order explanations of graph neural networks via relevant walks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. 7

[38] Y. Zhang, D. Defazio, and A. Ramesh, *RelEx: A Model-Agnostic Relational Model Explainer*. New York, NY, USA: Association for Computing Machinery, 2021, p. 1042–1049. [Online]. Available: https://doi.org/10.1145/3461702.3462562 7

[39] H. Yuan, J. Tang, X. Hu, and S. Ji, “XGNN: towards model-level explanations of graph neural networks,” *CoRR*, vol. abs/2006.02587, 2020. [Online]. Available: https://arxiv.org/abs/2006.02587 7

[40] H. Yuan, H. Yu, S. Gui, and S. Ji, “Explainability in graph neural networks: A taxonomic survey,” *CoRR*, vol. abs/2012.15445, 2020. [Online]. Available: https://arxiv.org/abs/2012.15445 7

[41] J. Mei and J. M. F. Moura, "Signal processing on graphs: Causal modeling of unstructured data," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 2077–2092, Apr. 2017. [Online]. Available: https://doi.org/10.1109/tsp.2016.2634543 13

[42] J. Mei, "Principal network analysis," Ph.D. dissertation, Carnegie Mellon University, 2018. 13

[43] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15. 14

[44] M. Püschel and J. M. F. Moura, "Algebraic signal processing theory: 1-D space," *IEEE Trans. Signal Processing*, vol. 56, no. 8-1, pp. 3586–3599, 2008. [Online]. Available: https://doi.org/10.1109/TSP.2008.925259 17

[45] J. Shi and J. M. F. Moura, "Graph signal processing: modulation, convolution, and sampling," *Computing Research Repository*, vol. abs/1912.06762, Dec. 2019. 19, 143

[46] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *2nd International Conference on Learning Representations, (ICLR) 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014. 22, 25

[47] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems (NeurIPS) 2016, December 5-10, 2016, Barcelona, Spain*, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 3837–3845. [Online]. Available: http://papers.nips.cc/paper/6081-convolutional-neural-networks-on-graphs-with-fast-localized-spectral-filtering 22

[48] J. Du, S. Zhang, G. Wu, J. M. F. Moura, and S. Kar, "Topology adaptive graph convolutional networks," *Computing Research Repository*, vol. abs/1710.10370, 2017. [Online]. Available: http://arxiv.org/abs/1710.10370 22, 23, 24, 34, 65, 126, 130

[49] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30*,

I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1024–1034. [Online]. Available: http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf 22

[50] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, (ICLR) 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ 22, 41, 44, 58, 75

[51] M. Cheung, J. Shi, O. Wright, Y. Jiang, and J. M. F. Moura, "Pooling in graph convolutional neural networks," in *53rd Asilomar Conference on Signals, Systems, and Computers*, 2019. 27, 143

[52] S. Verma and Z.-L. Zhang, "Hunt for the unique, stable, sparse and fast feature learning on graphs," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 88–98. [Online]. Available: http://papers.nips.cc/paper/6614-hunt-for-the-unique-stable-sparse-and-fast-feature-learning-on-graphs.pdf 28, 29

[53] S. Verma and Z. Zhang, "Graph capsule convolutional neural networks," *Computing Research Repository*, vol. abs/1805.08090, 2018. [Online]. Available: http://arxiv.org/abs/1805.08090 28, 29

[54] Z. Xinyi and L. Chen, "Capsule graph neural network," in *7th International Conference on Learning Representations, (ICLR) 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=Byl8BnRcYm 28, 29

[55] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective Classification in Network Data," *AI Magazine*, vol. 29, no. 3, 2008. 30

[56] "Node classification," https://paperswithcode.com/task/node-classification, 2021, accessed: 2021-09-30. 30

[57] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," *CoRR*, vol. abs/1903.02428, 2019. [Online]. Available: http://arxiv.org/abs/1903.02428 33, 69

[58] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 3538–3545. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16098 34, 35

[59] F. R. K. Chung, *Spectral Graph Theory*. American Mathematical Society, 1997. 35

[60] U. von Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007. [Online]. Available: https://doi.org/10.1007/s11222-007-9033-z 35

[61] K. Zhou, X. Huang, Y. Li, D. Zha, R. Chen, and X. Hu, "Towards deeper graph neural networks with differentiable group normalization," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 4917–4928. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/33dd6dba1d56e826aac1cbf23cdcca87-Paper.pdf 35

[62] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 3438–3445, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/5747 35

[63] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1ldO2EFPr 35

[64] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *CoRR*, vol. abs/1811.05868, 2018. [Online]. Available: http://arxiv.org/abs/1811.05868 36

[65] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, no. 2, pp. 127–163, Jul 2000. [Online]. Available: https://doi.org/10.1023/A:1009953814988 36

[66] C. L. Giles, K. D. Bollacker, and S. Lawrence, "Citeseer: An automatic citation indexing system," in *Proceedings of the Third ACM Conference on Digital Libraries*, ser. DL '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 89–98. [Online]. Available: https://doi.org/10.1145/276675.276685 36

[67] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, p. 93, Sep. 2008. [Online]. Available: https://ojs.aaai.org/index.php/aimagazine/article/view/2157 36

[68] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 43–52. [Online]. Available: https://doi.org/10.1145/2766462.2767755 36

[69] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications," *Phys. Rev. E*, vol. 64, p. 026118, Jul 2001. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.64.026118 45

[70] S. H. S. Duncan J. Watts, "Collective dynamics of 'small-world' networks," *NATURE*, vol. 393, 1998. [Online]. Available: https://www.nature.com/articles/30918.pdf 45

[71] A. Barrat and M. Weigt, "On the properties of small-world network models," *The European Physical Journal B*, vol. 13, no. 3, pp. 547–560, Jan. 2000. [Online]. Available: https://doi.org/10.1007/s100510050067 45

[72] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, vol. 74, pp. 47–97, Jan 2002. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.74.47 45

[73] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes, "Pseudofractal scale-free web," *Phys. Rev. E*, vol. 65, p. 066122, Jun 2002. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.65.066122 45

[74] "Node classification," https://paperswithcode.com/task/graph-classification, 2021, accessed: 2021-09-30. 61

[75] J. Shi, M. Cheung, J. Du, and J. M. F. Moura, "Classification with Vertex-Based Graph Convolutional Neural Networks," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, Oct 2018, pp. 752–756. 65, 143

[76] R. Sato, "A survey on the expressive power of graph neural networks," *Computing Research Repository*, vol. abs/2003.04078, 2020. 65, 66

[77] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-Lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, Nov. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2078187 66, 68

[78] J. Shi, M. Cheung, W. Summer, and J. M. F. Moura, "A dual approach to graph cnns," in *2020 54th Asilomar Conference on Signals, Systems, and Computers*, 2020, pp. 1467–1471. 66, 143

[79] A. K. Debnath, R. L. L. de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of Medicinal Chemistry*, vol. 34, no. 2, pp. 786–797, Feb. 1991. [Online]. Available: https://doi.org/10.1021/jm00106a046 68

[80] I. Schomburg, "BRENDA, the enzyme database: updates and major new developments," *Nucleic Acids Research*, vol. 32, no. 90001, pp. 431D–433, Jan. 2004. [Online]. Available: https://doi.org/10.1093/nar/gkh081 68

[81] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of Molecular Biology*, vol. 330, no. 4, pp. 771–783, Jul. 2003. [Online]. Available: https://doi.org/10.1016/s0022-2836(03)00628-4 68

[82] K. Riesen and H. Bunke, "IAM graph database repository for graph based pattern recognition and machine learning," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 287–297. 68

[83] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, Aug. 2007. [Online]. Available: https://doi.org/10.1007/s10115-007-0103-5 69

[84] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1365–1374. [Online]. Available: https://doi.org/10.1145/2783258.2783417 69

[85] B. Rozemberczki, O. Kiss, and R. Sarkar, "Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*. ACM, 2020, p. 3125–3132. 69

[86] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019. 69

[87] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. 70, 135

[88] T. Laurent and J. von Brecht, "Deep linear networks with arbitrary loss: All local minima are global," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 2902–2907. [Online]. Available: https://proceedings.mlr.press/v80/laurent18a.html 98

[89] M. Hardt and T. Ma, "Identity matters in deep learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26,*

2017, Conference Track Proceedings. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=ryxB0Rtxx 98

[90] S. Du and W. Hu, "Width provably matters in optimization for deep linear neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 1655–1664. [Online]. Available: https://proceedings.mlr.press/v97/du19a.html 98

[91] D. Zou, P. M. Long, and Q. Gu, "On the global convergence of training deep linear resnets," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HJxEhREKDH 98

[92] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 8580–8589. 98

[93] K. Xu, M. Zhang, J. Li, S. S. Du, K.-I. Kawarabayashi, and S. Jegelka, "How neural networks extrapolate: From feedforward to graph neural networks," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=UH-cmocLJC 98

[94] S. S. Du, K. Hou, R. R. Salakhutdinov, B. Poczos, R. Wang, and K. Xu, "Graph neural tangent kernel: Fusing graph neural networks with graph kernels," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alch'e-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/663fd3c5144fd10bd5ca6611a9a5b92d-Paper.pdf 98

[95] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 5680–5695, 2020. 98

[96] S. Zhang, M. Wang, S. Liu, P.-Y. Chen, and J. Xiong, "Fast learning of graph neural networks with guaranteed generalizability: One-hidden-layer case," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh,

Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 11 268–11 277. [Online]. Available: https://proceedings.mlr.press/v119/zhang20y.html 98, 142

[97] K. Xu, M. Zhang, S. Jegelka, and K. Kawaguchi, "Optimization of graph neural networks: Implicit acceleration by skip connections and more depth," in *ICML*, 2021, pp. 11 592–11 602. [Online]. Available: http://proceedings.mlr.press/v139/xu21k.html 98, 100, 142, 163

[98] P. Awasthi, A. Das, and S. Gollapudi, "A convergence analysis of gradient descent on graph neural networks," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 20 385–20 397. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/aaf2979785deb27864047e0ea40ef1b7-Paper.pdf 98, 142

[99] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014. 100, 163

[100] K. Kawaguchi, "On the theory of implicit deep learning: Global convergence with implicit layers," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=p-NZIuwqhI4 100, 163

[101] N. Kommenda and F. Hulley-Jones, "Covid vaccine tracker: when will a coronavirus vaccine be ready?" *The Guardian*, Nov. 2020. 126

[102] International Federation of Pharmaceutical Manufacturers and Associations, "The complex journey of a vaccine," Jul 2019. 126

[103] Y. Zhou, F. Wang, J. Tang, R. Nussinov, and F. Cheng, "Artificial intelligence in COVID-19 drug repurposing," *The Lancet Digital Health*, Sep. 2020. 126

[104] J. M. Levin, T. I. Oprea, S. Davidovich, T. Clozel, J. P. Overington, Q. Vanhaelen, C. R. Cantor, E. Bischof, and A. Zhavoronkov, "Artificial intelligence, drug repurposing and peer review," *Nature Biotechnology*, vol. 38, no. 10, pp. 1127–1131, Sep. 2020. 126

[105] K. Yang, K. Swanson, W. Jin, C. Coley, P. Eiden, H. Gao, A. Guzman-Perez, T. Hopper, B. Kelley, M. Mathea, A. Palmer, V. Settels, T. Jaakkola, K. Jensen, and R. Barzilay, "Analyzing learned molecular representations for property prediction," *Journal of Chemical Information and Modeling*, vol. 59, no. 8, pp. 3370–3388, 2019. 126, 134

[106] M. Cheung and J. M. F. Moura, "Graph neural networks for covid-19 drug discovery," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 5646–5648. 126, 143

[107] R. S. Bohacek, C. McMartin, and W. C. Guida, "The art and practice of structure-based drug design: A molecular modeling perspective," *Medicinal Research Reviews*, vol. 16, no. 1, pp. 3–50, 1996. 127

[108] P. Zhou, X.-L. Yang, X.-G. Wang, B. Hu, L. Zhang, W. Zhang, H.-R. Si, Y. Zhu, B. Li, C.-L. Huang, H.-D. Chen, J. Chen, Y. Luo, H. Guo, R.-D. Jiang, M.-Q. Liu, Y. Chen, X.-R. Shen, X. Wang, X.-S. Zheng, K. Zhao, Q.-J. Chen, F. Deng, L.-L. Liu, B. Yan, F.-X. Zhan, Y.-Y. Wang, G.-F. Xiao, and Z.-L. Shi, "A pneumonia outbreak associated with a new coronavirus of probable bat origin," *Nature*, vol. 579, no. 7798, pp. 270–273, Mar 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2012-7 127

[109] T. Hoffmann and M. Gastreich, "The next level in chemical space navigation: going far beyond enumerable compound libraries," *Drug Discovery Today*, vol. 24, no. 5, pp. 1148–1156, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1359644618304471 127

[110] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of Computer-Aided Molecular Design*, vol. 30, no. 8, pp. 595–608, Aug. 2016. 128

[111] Y.-C. Lo, S. E. Rensi, W. Torng, and R. B. Altman, "Machine learning in chemoinformatics and drug discovery," *Drug Discovery Today*, vol. 23, no. 8, pp. 1538 – 1546, 2018. 128

[112] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 1263–1272. 130

[113] B. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA) - Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0005279575901099 132

[114] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (MCC) over f1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, Jan. 2020. [Online]. Available: https://doi.org/10.1186/s12864-019-6413-7 132

[115] D. L. Group, "Main protease structure and xchem fragment screen," https://www.diamond.ac.uk/covid-19/for-scientists/Main-protease-structure-and-XChem.html, accessed: 2021-01-01. 133

[116] F. Touret, M. Gilles, K. Barral, A. Nougairède, E. Decroly, X. de Lamballerie, and B. Coutard, "In vitro screening of a fda approved chemical library reveals potential inhibitors of sars-cov-2 replication," *bioRxiv*, 2020. [Online]. Available: https://www.biorxiv.org/content/early/2020/04/05/2020.04.03.023846 133

[117] B. Ellinger, D. Bojkova, A. Zaliani, J. Cinatl, C. Claussen, S. Westhaus, O. Keminer, J. Reinshagen, M. Kuzikov, M. Wolf, G. Geisslinger, P. Gribbon, and S. Ciesek, "A SARS-CoV-2 cytopathicity dataset generated by high-content screening of a large drug repurposing collection," *Scientific Data*, vol. 8, no. 1, feb 2021. 134

[118] B. I. The Drug Repurposing Hub, "Evaluation set for sars-cov 3cl protease," https://clue.io/repurposing, 2020. 134

[119] T. S. R. I. M. S. Center, "Pubchem bioassay record for aid 1706," https://pubchem.ncbi.nlm.nih.gov/bioassay/1706, 2020. 134

[120] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 115–123. [Online]. Available: https://proceedings.mlr.press/v28/bergstra13.html 134

[121] F. Wu, T. Zhang, A. H. d. Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *36th International Conference on Machine Learning (ICML)*, 2019. [Online]. Available: https://arxiv.org/abs/1902.07153v1 143

[122] Y. Li and J. M. F. Moura, "Forecaster: A graph transformer for forecasting spatial and time-dependent data," in *ECAI 2020 : 24th European Conference on Artificial Intelligence*, vol. abs/1909.04019, Santiago de Compostela, Spain, June 2020. [Online]. Available: http://arxiv.org/abs/1909.04019 144

[123] K. Oono and T. Suzuki, "Graph Neural Networks Exponentially Lose Expressive Power for Node Classification," *International Conference on Learning Representations 2020*, 2020. 144

Appendix A: Graph Classification Training Convergence

We also consider the task training convergence for graph classification. The main difference is the requirement of the aggregation layer that complicates the derivation (See Section 3.4). More specifically, the orthogonal projection of the aggregated output on $I_{C_y} \otimes \mathbf{A}^H X$ does not simplify to $\widehat{Y}$ (as shown for node classification in Section 6.2.1.4). We provide a sketch of required steps here, and leave the complete proof for future work.

The chapter is separated into two parts. We start by formally defining the problem of training convergence in Section A.1. Then we discuss 2 approaches for the convergence for H-layer, 1-degree TAGCN in Section A.2.

A.1 Problem Definition

Following the same notations as Section 5.1, we consider the supervised learning setting with the following observations in the forms of graphs $\mathcal{G}_s = (\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_N)$, and labels $\mathbf{Y}_s = (\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_N)$. We are given a subset of labels for training $\{\mathbf{Y}_i \mid i \in D_{\text{training}}\}$, and the task is to predict the rest $\{\mathbf{Y}_i \mid i \in D_{\text{testing}}\}$, where D_{training} and D_{testing} correspond to the train and test sets, respectively.

Each graph $\mathcal{G}_i$ consists of a set of nodes V_i with $|V_i| = N_i$, signal $\mathbf{X}_i \in \mathbb{R}^{N_i \times C_0}$ on the nodes (where C_0 is the number of features on each node), a set of edges $\mathcal{E}_i$, and an adjacency matrix $\mathbf{A}_i$. $[\mathbf{A}_i]_{jk} = 0$ unless there is an edge $e = (j,k)$ connecting node j and node k. Generally, the number of features is the same for all graphs.

Given a neural network function $f(\mathbf{X}_i, \mathbf{A}_i)$, we use a non-negative real-valued loss function to model the prediction error of each sample:

$$L(f(\mathbf{X}_i, \mathbf{A}_i), \mathbf{Y}_i) \tag{A.1}$$

For GNNs, the function is a set of convolutional layers. For TAGCN, each convolutional layer is:

$$\mathbf{X}^{(\ell+1)} = \sigma\left(\sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}^{(\ell)} \mathbf{W}_k^{(\ell)}\right), \tag{3.2}$$

where $\mathbf{W}_k^{(\ell)} \in \mathbb{R}^{C_\ell \times C_{\ell+1}}$, $\mathbf{X}^{(\ell)} \in \mathbb{R}^{N \times C_\ell}$, and the input signal $\mathbf{X} = \mathbf{X}^{(0)}$. For training convergence, we prove cases without nonlinearity σ:

$$\mathbf{X}^{(\ell+1)} = \sum_{k=0}^{K} \mathbf{A}^k \mathbf{X}^{(\ell)} \mathbf{W}_k^{(\ell)}, \tag{6.2}$$

and starting with the case of no polynomial and no residual connections:

$$x^{(\ell+1)} = \mathbf{A}^k \mathbf{X}^{(\ell)} \mathbf{W}_k^{(\ell)}. \tag{6.3}$$

After graph convolutions, we apply the aggregation so that graphs of different sizes and orders are comparable. We choose the mean as it is the most popular aggregation that performs well:

$$f(\mathbf{X}, \mathbf{A}, \mathbf{W}, \mathbf{B}) = \frac{1}{N} \sum_{j=1}^{N} \mathbf{X}_j^{(H)} \mathbf{B}, \tag{A.2}$$

where j represents iterates through row index of $\mathbf{X}$, so $f(\mathbf{X}, \mathbf{A}, \mathbf{W}, \mathbf{B}) \in \mathbb{R}^{C_Y}$ with $\mathbf{B} \in \mathbb{R}^{C_H \times C_Y}$.

Definition 4. (Global minimum) L^* is the global minimum value of GNN f

$$L^* = \inf_{\mathbf{W}, \mathbf{B}} L(f(\mathbf{X}, \mathbf{A}, \mathbf{W}, \mathbf{B}), \mathbf{Y}), \tag{A.3}$$

where for graph classification, the loss is over all the graphs in training.

The goal is to study the convergence

$$\| \lim_{t \to \infty} L(\mathbf{W}_t, \mathbf{B}_t) - L^* \|$$

and to analyze the rate of convergence in terms of number of layers and degrees of the polynomial filters.

A.2 Convergence Theory

In this section, we sketch out the steps for global convergence for TAGCN for graph classification. Similar to node classification, we follow previous related work on this topic [97, 99, 100] and extend them to the TAGCN architecture. We only consider H-layer 1-degree case.

We consider two approaches in Section A.2.1 and Section A.2.2, respectively. For our first approach, we assume that the graphs all have the same size and order, and we apply a weight matrix that convert the output of the graph convolutions to the same dimension of Y for classification. For our second approach, we incorporate the aggregation into the loss function, allowing us to reuse the first 3 steps of the derivation for node classification.

A.2.1 Approach # 1: Graphs with Same Size and Order

For this approach, we assume all the graphs have the same size and that the order is significant. By order, we mean that indexing of $\mathbf{X}$ (and hence $\mathbf{A}$) is fixed. This allows us to apply a linear layer to convert the number of nodes to the number of labels in Y. Instead of (A.2), we have simply,

$$f(\mathbf{X}, \mathbf{A}, \mathbf{W}, \mathbf{B}) = \mathbf{B}\mathbf{X}^{(H)} \tag{A.4}$$

From (6.3), we have $\mathbf{X}^{(\ell)} = \mathbf{A}\mathbf{X}^{(\ell-1)}\mathbf{W}^{(\ell-1)}$ (ignoring the non-linearity and the residual connection). Combining these equations, we derive $\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{B}]} \in \mathbb{R}^{C_y N \times C_y C_H}$:

$$\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{B}]} = \frac{\partial}{\partial \operatorname{vec}[\mathbf{B}]} \operatorname{vec}[\mathbf{B}\mathbf{X}^{(H)}] \tag{A.5}$$

$$= \frac{\partial}{\partial \operatorname{vec}[\mathbf{B}]} [\mathbf{X}^{(H)^\top} \otimes I_{C_y}] \operatorname{vec}[\mathbf{B}]$$

$$= [\mathbf{X}^{(H)^\top} \otimes I_{C_y}] \in \mathbb{R}^{1C_y \times NC_y} \tag{A.6}$$

With this, we derive the formula for $\nabla_{\mathbf{B}} L(\mathbf{B}, \mathbf{W}) \in \mathbb{R}^{C_H \times C_y}$:

$$\frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \operatorname{vec}[\mathbf{B}]} = \frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{B}]} = \frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} [\mathbf{X}^{(H)^\top} \otimes I_{C_y}]$$

Thus,

$$\nabla_{\operatorname{vec}[\mathbf{B}]} L(\mathbf{B}, \mathbf{W}) = \left(\frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \operatorname{vec}[\mathbf{B}]} \right)^\top$$

$$= [\mathbf{X}^{(H)^\top} \otimes I_{C_y}]^\top \left(\frac{\partial L(\mathbf{B}, \mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \right)^\top$$

$$\begin{aligned}
&= [\mathbf{X}^{(H)} \otimes I_{C_y}] \operatorname{vec}\left[\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}}\right] \\
&= \operatorname{vec}\left[\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}}\mathbf{X}^{(H)^\top}\right] \in \mathbb{R}^{C_Y C_N}.
\end{aligned}$$

Therefore,

$$\nabla_{\mathbf{B}} L(\mathbf{B},\mathbf{W}) = \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}}\mathbf{X}^{(H)^\top} \in \mathbb{R}^{C_Y \times N}. \tag{A.7}$$

Next, we derive $\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \in \mathbb{R}^{C_y N \times C_l C_{l+1}}$:

$$\begin{aligned}
\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} &= \frac{\partial \operatorname{vec}[\mathbf{B}\mathbf{X}^{(H)}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \\
&= \frac{\partial}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]}[I_{C_H} \otimes \mathbf{B}] \operatorname{vec}[(\mathbf{X}^{(H)})]] \\
&= [I_{C_H} \otimes \mathbf{B}] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})]]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \\
&= [I_{C_H} \otimes \mathbf{B}] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} \frac{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \\
&= [I_{C_H} \otimes \mathbf{B}] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} \frac{\partial \operatorname{vec}[\mathbf{A}\mathbf{X}^{(\ell)}\mathbf{W}^{(\ell)}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \\
&= [I_{C_H} \otimes \mathbf{B}] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} \frac{\partial [I_{C_l} \otimes \mathbf{A}\mathbf{X}^{(\ell)}] \operatorname{vec}[\mathbf{W}^{(\ell)}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} \\
&= [I_{C_H} \otimes \mathbf{B}] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} [I_{C_l} \otimes \mathbf{A}\mathbf{X}^{(\ell)}]
\end{aligned} \tag{A.8}$$

From (6.3), we have $\mathbf{X}^{(\ell)} = \mathbf{A}\mathbf{X}^{(\ell-1)}\mathbf{W}^{(\ell-1)}$, so

$$\operatorname{vec}[(\mathbf{X}^{(H)})] = \operatorname{vec}[\mathbf{A}\mathbf{X}^{(H-1)}\mathbf{W}^{(H-1)}] = (\mathbf{W}^{(H-1)^\top} \otimes \mathbf{A}) \operatorname{vec}[\mathbf{X}^{(H-1)}] \tag{A.9}$$

and

$$\operatorname{vec}[(\mathbf{X}^{(\ell)})] = \operatorname{vec}[\mathbf{A}\mathbf{X}^{(\ell-1)}\mathbf{W}^{(\ell-1)}] = (\mathbf{W}^{(\ell-1)^\top} \otimes \mathbf{A}) \operatorname{vec}[\mathbf{X}^{(\ell-1)}]. \tag{A.10}$$

By recursively applying (A.10), we get

$$\begin{aligned}
\operatorname{vec}[(\mathbf{X}^{(H)})] &= (\mathbf{W}^{(H-1)^\top} \otimes \mathbf{A})(\mathbf{W}^{(H-2)^\top} \otimes \mathbf{A}) \cdots (\mathbf{W}^{(\ell+1)^\top} \otimes \mathbf{A}) \operatorname{vec}[\mathbf{X}^{(\ell+1)}] \\
&= (\mathbf{W}^{(H-1)^\top} \cdots \mathbf{W}^{(\ell+1)^\top} \otimes \mathbf{A}^{H-l-1}) \operatorname{vec}[\mathbf{X}^{(\ell+1)}].
\end{aligned}$$

Hence,

$$\frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} = \mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top} \otimes \mathbf{A}^{H-l-1} \tag{A.11}$$

Combining (A.8) and (A.11) gives:

$$\begin{aligned}
\frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} &= [I_{C_H} \otimes \mathbf{B}] \frac{\partial \operatorname{vec}[(\mathbf{X}^{(H)})]}{\partial \operatorname{vec}[\mathbf{X}^{(\ell+1)}]} [I_{C_l} \otimes \mathbf{A}\mathbf{X}^{(\ell)}] \\
&= [I_{C_H} \otimes \mathbf{B}][\mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top} \otimes \mathbf{A}^{H-l-1}][I_{C_l} \otimes \mathbf{A}\mathbf{X}^{(\ell)}] \\
&= \mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top} \otimes \mathbf{B}\mathbf{A}^{H-l}\mathbf{X}^{(\ell)} \in \mathbb{R}^{C_y \times C_l C_{l+1}}.
\end{aligned} \tag{A.12}$$

With this, we derive the formula of $\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{B}, \mathbf{W}) \in \mathbb{R}^{C_l \times C_{l+1}}$:

$$\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} = \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \frac{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]}{\partial \operatorname{vec}[\mathbf{W}^{(\ell)}]} = \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} [\mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top} \otimes \mathbf{B}\mathbf{A}^{H-l}\mathbf{X}^{(\ell)}].$$

Thus,

$$\begin{aligned}
\nabla_{\operatorname{vec}[\mathbf{W}^{(\ell)}]} L(\mathbf{B},\mathbf{W}) &= [\mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top} \otimes \mathbf{B}\mathbf{A}^{H-l}\mathbf{X}^{(\ell)}]^\top \left(\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \right)^\top \\
&= [(\mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top})^\top \otimes [\mathbf{B}\mathbf{A}^{H-l}\mathbf{X}^{(\ell)}]^\top] \left(\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \right)^\top \\
&= [(\mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top})^\top \otimes \mathbf{X}^{(\ell)^\top} \mathbf{A}^{H-l^\top} \mathbf{B}^\top] \left(\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \operatorname{vec}[\widehat{\mathbf{Y}}]} \right)^\top \\
&= \operatorname{vec}\left[\mathbf{X}^{(\ell)^\top} (\mathbf{A}^{H-l})^\top \mathbf{B}^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top} \right] \in \mathbb{R}^{C_l C_{l+1}} \\
&= \operatorname{vec}\left[(\mathbf{B}\mathbf{A}^{H-l}\mathbf{X}^{(\ell)})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top} \right] \in \mathbb{R}^{C_l C_{l+1}}
\end{aligned}$$

Therefore,

$$\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{B},\mathbf{W}) = (\mathbf{B}\mathbf{A}^{H-l}\mathbf{X}^{(\ell)})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top} \in \mathbb{R}^{C_l \times C_{l+1}}. \tag{A.13}$$

Expanding (A.7):

$$\begin{aligned}
\nabla_{\mathbf{B}} L(\mathbf{B},\mathbf{W}) &= \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{X}^{(H)^\top} \\
&= \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} (\mathbf{A}\mathbf{X}^{(H-1)}\mathbf{W}^{(H-1)})^\top
\end{aligned}$$

$$= \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} (\mathbf{A}^H \mathbf{X} \mathbf{W}^{(0)} \dots \mathbf{W}^{(H-1)})^\top$$
$$= \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} (\mathbf{W}^{(0)} \dots \mathbf{W}^{(H-1)})^\top (\mathbf{A}^H \mathbf{X})^\top$$

Expanding (A.13):

$$\begin{aligned}\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{B},\mathbf{W}) &= (\mathbf{B}\mathbf{A}^{H-l}\mathbf{X}^{(\ell)})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top} \\ &= (\mathbf{B}\mathbf{A}^{H-l+l}\mathbf{X}\mathbf{W}^{(0)} \dots \mathbf{W}^{(\ell-1)})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top} \\ &= (\mathbf{W}^{(0)} \dots \mathbf{W}^{(\ell-2)}\mathbf{W}^{(\ell-1)})^\top (B\mathbf{A}^H\mathbf{X})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}} \mathbf{W}^{(H-1)^\top} \dots \mathbf{W}^{(\ell+1)^\top}\end{aligned}$$

The next step is to consider the dynamics induced in the space of $W^{(0)}W^{(1)}W^{(2)} \dots W^{(\ell)}$ as well as B. They need to be considered separately as $\mathbf{B}$ is on the left hand side, while the $\mathbf{W}$'s are on right hand side.

A.2.2 Approach #2: Modification of Squared Loss

In this section, we consider the sum aggregation in (A.2) and incorporate it into the loss. This allows us to use the same initial steps as node classification described in Section 6.2.1, but without the selection of nodes ($*\mathcal{I}$). More specifically, they are 1) compute the gradients of the loss with respect to the parameter $\nabla_{\mathbf{W}^{(\ell)}} L(\mathbf{W})$ in Section 6.2.1.1, 2) analyze the dynamics induced in $\mathbf{W}^{(0)}\mathbf{W}^{(1)} \dots \mathbf{W}^{(H-1)}$ in Section 6.2.1.2, and 3) analyze the dynamics induced in the loss value $L(\mathbf{W})$ in Section 6.2.1.3. The last step (proof with square loss) is sketched below.

We complete the proof by aggregating the nodes and using squared loss. With $\widehat{\mathbf{Y}} = f(\mathbf{X},\mathbf{B},\mathbf{W})$, $L(f(\mathbf{X},\mathbf{B},\mathbf{W})\mathbf{Y}) = \|\frac{1}{N}(\sum_{i=1}^N \widehat{\mathbf{Y}}_i) - \mathbf{Y}\|_F^2$ where $\widehat{\mathbf{Y}}_i \in \mathbb{R}^{1\times C_y}$ corresponds to the value of node i (row i). So we have:

$$\frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}_i} = \frac{\partial}{\partial \widehat{\mathbf{Y}}_i} \|\frac{1}{N}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \mathbf{Y}\|_F^2 = 2[\frac{1}{N}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \mathbf{Y}]\frac{1}{N} = \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \in \mathbb{R}^{1\times C_y},$$

hence,

$$\frac{\partial L(\mathbb{B},\mathbb{W})}{\partial \widehat{\mathbb{Y}}} = \begin{bmatrix} \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \vdots \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \end{bmatrix} \in \mathbb{R}^{N\times C_y},$$

With these, the rest of the steps can follow similar to Section 6.2.1.4.

$$\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})] = \mathrm{vec}\left[(\mathbf{A}^H\mathbf{X})^\top \frac{\partial L(\mathbf{B},\mathbf{W})}{\partial \widehat{\mathbf{Y}}}\right] = 2\,\mathrm{vec}\left[(\mathbf{A}^H\mathbf{X})^\top \begin{bmatrix} \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \vdots \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \end{bmatrix}\right]$$

$$= 2[I_{C_y} \otimes (\mathbf{A}^H\mathbf{X})^\top]\,\mathrm{vec}\begin{bmatrix} \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \vdots \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \end{bmatrix}.$$

Hence,

$$\|\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})]\|_2^2 = 4\,\mathrm{vec}\begin{bmatrix} \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \vdots \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N}\mathbf{X} \end{bmatrix}^\top [I_{C_y} \otimes (\mathbf{A}^H\mathbf{X})(\mathbf{A}^H\mathbf{X})^\top] \tag{A.14}$$

$$\mathrm{vec}\begin{bmatrix} \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \vdots \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N}\mathbf{X} \end{bmatrix} \tag{A.15}$$

Plugging in (A.14) into (6.32),

$$\frac{d}{dt}L(\mathbf{B},\mathbf{W}) \leq -\lambda_{\mathbf{B},\mathbf{W}}\|\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})]\|_2^2 - \sum_{i=0}^{H-1}\left\|J_{(i,H)}\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})]\right\|_2^2$$

$$= -4\lambda_{\mathbf{B},\mathbf{W}}\,\mathrm{vec}\begin{bmatrix} \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \vdots \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N}\mathbf{X} \end{bmatrix}^\top [I_{C_y} \otimes (\mathbf{A}^H\mathbf{X})(\mathbf{A}^H\mathbf{X})^\top]$$

$$\mathrm{vec}\begin{bmatrix} \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \vdots \\ \frac{2}{N^2}(\sum_{j=1}^N \widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N}\mathbf{X} \end{bmatrix} - \sum_{i=0}^{H-1}\left\|J_{(i,H)}\,\mathrm{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})]\right\|_2^2$$

$$= -4\lambda_{\mathbf{B},\mathbf{W}} \operatorname{vec}\begin{bmatrix} \frac{2}{N^2}(\sum_{j=1}^{N}\widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \frac{2}{N^2}(\sum_{j=1}^{N}\widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \vdots \\ \frac{2}{N^2}(\sum_{j=1}^{N}\widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N}\mathbf{X} \end{bmatrix}^{\top} \left[I_{C_y} \otimes \mathbf{G}_H\mathbf{G}_H^{\top}\right] \operatorname{vec}\begin{bmatrix} \frac{2}{N^2}(\sum_{j=1}^{N}\widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \frac{2}{N^2}(\sum_{j=1}^{N}\widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \vdots \\ \frac{2}{N^2}(\sum_{j=1}^{N}\widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N}\mathbf{X} \end{bmatrix}$$
$$- \sum_{i=0}^{H-1} \left\| J_{(i,H)} \operatorname{vec}[\nabla_{(H)} L(\mathbf{B},\mathbf{W})] \right\|_2^2$$

where $\mathbf{G}_H := (\mathbf{A}^H)\mathbf{X}$.

In order to simplify this, we need to project

$$\operatorname{vec}\begin{bmatrix} \frac{2}{N^2}(\sum_{j=1}^{N}\widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \frac{2}{N^2}(\sum_{j=1}^{N}\widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N} \\ \vdots \\ \frac{2}{N^2}(\sum_{j=1}^{N}\widehat{\mathbf{Y}}_j) - \frac{2\mathbf{Y}}{N}\mathbf{X} \end{bmatrix}$$

onto

$$\mathbf{P}_{I_{C_y} \otimes \mathbf{G}_H} \in \mathbb{R}^{C_y N \times C_y N}.$$

This projection does not work out nicely as the summation is no longer a multiple of $\mathbf{G} = \mathbf{A}^H\mathbf{X}$.