# Graph Neural Networks with Adaptive Frequency Response Filter

Yushun Dong
University of Virginia
yd6eb@virginia.edu

Kaize Ding
Arizona State University
kding9@asu.edu

Brian Jalaian
Army Research Laboratory
brian.a.jalaian.civ@mail.mil

Shuiwang Ji
Texas A&M University
sji@tamu.edu

Jundong Li
University of Virginia
jundong@virginia.edu

## ABSTRACT

Graph Neural Networks have recently become a prevailing paradigm for various high-impact graph learning tasks. Existing efforts can be mainly categorized as spectral-based and spatial-based methods. The major challenge for the former is to find an appropriate graph filter to distill discriminative information from input signals for learning. Recently, attempts such as Graph Convolutional Network (GCN) leverage Chebyshev polynomial truncation to seek an approximation of graph filters and bridge these two families of methods. It has been shown in recent studies that GCN and its variants are essentially employing fixed low-pass filters to perform information denoising. Thus their learning capability is rather limited and may over-smooth node representations at deeper layers. To tackle these problems, we develop a novel graph neural network framework AdaGNN with a well-designed adaptive frequency response filter. At its core, AdaGNN leverages a simple but elegant trainable filter that spans across multiple layers to capture the varying importance of different frequency components for node representation learning. The inherent differences among different feature channels are also well captured by the filter. As such, it empowers AdaGNN with stronger expressiveness and naturally alleviates the over-smoothing problem. We empirically validate the effectiveness of the proposed framework on various benchmark datasets. Theoretical analysis is also provided to show the superiority of the proposed AdaGNN. The implementation of AdaGNN is available at https://github.com/yushundong/AdaGNN.

**ACM Reference Format:**
Yushun Dong, Kaize Ding, Brian Jalaian, Shuiwang Ji, and Jundong Li. 2018. Graph Neural Networks with Adaptive Frequency Response Filter. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

## 1 INTRODUCTION

Graph Neural Networks (GNNs) have demonstrated remarkable performance in a wide spectrum of graph learning tasks, e.g., node classification [15, 20], link prediction [31, 50], and recommendation [11, 37]. The main intuition of GNNs is that they stack multiple layers of neural network primitives to learn high-level node feature representations, aiming at addressing various learning tasks in an end-to-end manner. GNNs are deeply influenced by the architecture design of convolutional neural networks (CNNs) for grid-like data such as images and texts [17, 18, 21, 44] and are extended to handle non-Euclidean graph data. In essence, existing GNNs are mainly divided into two main streams: spectral-based and spatial-based methods [39]. With deep roots in graph signal processing [34] and spectral graph theory [8], spectral-based methods [4, 9, 20, 22] define convolution operations in the spectral domain based on graph Fourier transform and thus bear a solid mathematical foundation. For spatial-based methods, the convolution operations are defined in the spatial domain and act as a message-passing process [1, 13, 29]. Specifically, for each node, the convolution operations aggregate and transform information from its neighborhoods when learning its feature representation.

The seminal work of Graph Convolutional Network (GCN) [20] bridges the gap between these two families of algorithms. As a localized first-order approximation of spectral graph convolution [9], GCN can also be interpreted as a spatial-based method with a clear meaning of node localization, thus inspiring a lot of follow-up improvements [1, 13, 15, 22, 25, 36, 38, 40, 49], especially in the spatial domain. However, the fundamental studies and improvements of GCN from the spectral perspective is rather limited. Until fairly recently, studies have shown that the frequency response of GCN acts as a band-stop filter, and the convolution operation corresponds to a fixed low-pass filter due to high frequencies sparsity [30], implying that more information is captured within low frequencies and the effects of high-frequency components are reduced [38]. Despite that, real-world graphs are much more complex than we can imagine, and mixture of different frequencies may either benefit or degrade the final performance. The lowest frequency does not necessarily contain the most important information; while high-frequency components may also encode useful information that is beneficial for learning [3, 7]. In this regard, simply using a fixed low-pass filter cannot well capture the varying importance of different frequency components, thus limiting the expressiveness of learned representations and yielding suboptimal learning performance. Additionally, we show in this paper that in the limit, any filter satisfying certain conditions leads to another fundamental limitation of GCN and its variants – the over-smoothing problem [24]. It refers to the phenomenon that node feature representations converge to similar values at deeper layers, thus nodes cannot be easily distinguished.

To tackle the aforementioned problems, in this work we overcome the fundamental limitations of GCN and its variants from the spectral perspective. Specifically, we develop a novel GNN framework AdaGNN with an adaptive frequency response filter, which can adaptively adjust the importance of different frequency components for spectral convolution. Intuitively, to control the varying effects of different frequency components, a straightforward solution is to allocate a learnable parameter for each frequency component. However, this strategy requires expensive eigendecomposition [4] and its large parameter complexity will make the model prone to overfitting especially when the training data is limited. In this regard, we propose a simple but elegant solution to assign a single parameter for each feature channel at each layer, based upon which we stack multiple layers for a more powerful filter. In a nutshell, the main contributions of this paper can be summarized as follows:

- We systematically examine the fundamental limitations of fixed low-pass filters of GCN and its variants from the perspective of spectral domain.
- We develop a novel graph neural network framework named AdaGNN that can capture the varying importance of different frequency components for node representation learning. The core of this framework is a simple but elegant trainable filter that spans across multiple layers with a single parameter for each feature channel at each layer. The developed GNN framework does not involve expensive eigendecomposition and its parameter complexity is comparable to the lightweight GCN model such as SGC [38].
- We provide theoretical analysis for the proposed framework. Firstly, we show that the filter of prevalent GCN models (e.g., GCN and mean aggregator of GraphSage [15]) can be considered as a special case of our proposed adaptive filter at each layer. Secondly, we provide both spectral and spatial interpretation of the proposed framework. Thirdly, we also prove that any filter satisfying certain conditions will inevitably encounter over-smoothness in deeper structure, while the proposed filter can naturally alleviate this issue.
- We conduct comprehensive experiments on benchmark graph datasets with different properties. The empirical evaluations demonstrate that the proposed AdaGNN not only learns more powerful node representations for the node classification task but also greatly alleviate the over-smoothing problems when the architecture goes deeper.

## 2 THE PROPOSED FRAMEWORK – ADAGNN

### 2.1 Notations and Preliminaries

**Notations.** We define an undirected graph as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, ..., v_N\}$ and $\mathcal{E}$ denote the set of nodes and edges, respectively. Let $\mathbf{A} \in \mathbb{R}^{N\times N}$ be the adjacency matrix of the graph such that $\mathbf{A}_{i,j} = 1$ if $v_j \in \mathcal{N}(v_i)$, otherwise $\mathbf{A}_{i,j} = 0$. Here $\mathcal{N}(.)$ denotes the one-hop neighbor set of a node. The Laplacian matrix of the graph is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D} = \text{diag}(d_1, ..., d_N)$ is the diagonal degree matrix ($d_i = \sum_j \mathbf{A}_{i,j}$). Then the symmetric normalized Laplacian matrix and the random-walk normalized Laplacian matrix are defined as $\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$ and $\mathbf{L}_{rw} = \mathbf{D}^{-1}\mathbf{L}$, respectively. Besides, feature matrix $\mathbf{X} \in \mathbb{R}^{N\times F}$ is utilized to describe properties of nodes, where $\mathbf{x}_j$ (column of $\mathbf{X}$) represents $j$-th feature channel of $\mathbf{X}$ and $F$ denotes the number of feature channels.

**Graph Filters.** The main idea of spectral-based GNN methods is to define graph filters based on graph signal processing [34]. Specifically, symmetric normalized Laplacian can be factored as $\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$. Here $\mathbf{U} \in \mathbb{R}^{N\times N} = [\mathbf{u}_1, ..., \mathbf{u}_N]$, where $\mathbf{u}_i \in \mathbb{R}^N$ denotes the $i$-th eigenvector of $\mathbf{L}_{sym}$ and $\mathbf{\Lambda} = \text{diag}(\lambda_1, , ..., \lambda_n)$ is the corresponding eigenvalue matrix. Let $\mathbf{x} \in \mathbb{R}^N$ be an one-channeled input signal of all nodes, then the Graph Fourier transform and inverse Fourier transform can be defined as $\hat{\mathbf{x}} = \mathscr{F}(\mathbf{x}) = \mathbf{U}^T\mathbf{x}$ and $\mathbf{x} = \mathscr{F}^{-1}(\hat{\mathbf{x}}) = \mathbf{U}\hat{\mathbf{x}}$ respectively, where $\hat{\mathbf{x}}$ is the Fourier transformed graph signal. Graph convolution of the input signal $\mathbf{x}$ with filter $\mathbf{g} = \text{diag}(\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^N$ is defined as $\mathbf{x} *_G \mathbf{g} = \mathbf{U}\mathbf{g}(\mathbf{\Lambda})\mathbf{U}^T\mathbf{x}$. A vast majority of existing works such as GCN and SGC [20, 38] use a fixed low-pass filter for the graph convolution operation while recent studies [30, 38] have shown that if the input signal is repeatedly convolved with the fixed low-pass filter, its high-frequency components will be greatly weakened and the learning performance is dominated by the low-frequency components, resulting in the well-known over-smoothing problem.

### 2.2 Adaptive Frequency Response Filtering

In image signal processing, the Laplacian kernel is widely used to capture high-frequency edge information for various tasks such as image sharpening and blurring [14, 16]. As its counterpart in graph signal processing, we can multiply the graph Laplacian matrix $\mathbf{L}$ with the input graph signal $\mathbf{x} \in \mathbb{R}^N$ (i.e., $\mathbf{h} = \mathbf{L}\mathbf{x}$) to characterize its high-frequency components – the frequencies that carry sharply varying signal information across edges of graph. Meanwhile, as shown in recent studies [38, 41], the essence of GCN and its variants are the low-pass filter which smoothes the feature representations of the current node and its neighbors to make them similar. As such, we can highlight the low-frequency components of input signal $\mathbf{x}$ by setting $\mathbf{z} = \mathbf{x} - \mathbf{L}\mathbf{x}$, i.e., subtracting the term $\mathbf{L}\mathbf{x}$ which emphasizes more on high-frequency components from the input signal $\mathbf{x}$. In fact, this formulation is well aligned with the convolution operation in GCN [20] if we replace $\mathbf{L}$ with the symmetric normalized Laplacian matrix $\tilde{\mathbf{L}}_{sym}$ which is derived from the self-loop augmented adjacency matrix $\tilde{\mathbf{A}}$:

$$\mathbf{z} = \mathbf{x} - \tilde{\mathbf{L}}_{sym}\mathbf{x} = (\mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}}(\tilde{\mathbf{D}} - \tilde{\mathbf{A}})\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{x} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{x}, \quad (1)$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is used to avoid numerical instability issues [20] and $\tilde{\mathbf{D}}$ is the degree matrix from $\tilde{\mathbf{A}}$. Other than that, we can also use the random-walk normalized Laplacian matrix $\tilde{\mathbf{L}}_{rw}$ from $\tilde{\mathbf{A}}$. Here, we use $\tilde{\mathbf{L}}$ to denote the general formulation and it can be instantiated as $\tilde{\mathbf{L}}_{sym}$ and $\tilde{\mathbf{L}}_{rw}$.

The above operation corresponds to the fixed low-pass filter in the spectral domain, where higher weights are specified for low-frequency components. However, in practice, low-frequency components may not always be useful, and high-frequency can also provide complementary insights for learning [3, 7], especially when the label information is not smooth across edges. In this regard, it is

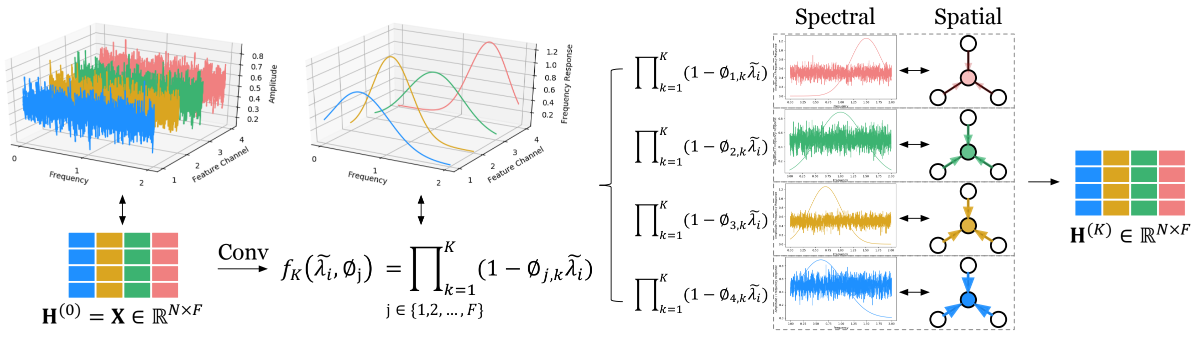

Frequency
Feature Channel
Amplitude
Frequency Response
$\mathbf{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{N\times F}$
Conv
$f_K(\tilde{\lambda}_i, \phi_j) = \prod_{k=1}^{K}(1-\phi_{j,k}\tilde{\lambda}_i)$
$j \in \{1,2,\ldots,F\}$
Spectral
Spatial
$\prod_{k=1}^{K}(1-\phi_{1,k}\tilde{\lambda}_i)$
$\prod_{k=1}^{K}(1-\phi_{2,k}\tilde{\lambda}_i)$
$\prod_{k=1}^{K}(1-\phi_{3,k}\tilde{\lambda}_i)$
$\prod_{k=1}^{K}(1-\phi_{4,k}\tilde{\lambda}_i)$
$\mathbf{H}^{(K)} \in \mathbb{R}^{N\times F}$

**Figure 1: An illustration of the proposed AdaGNN from the spectral perspective. The convolution operation across $K$ layers is equivalent to applying the filter $\prod_{k=1}^{K}(1-\phi_{j,k}\tilde{\lambda}_i)$ to the $j$-th channel of input signal $\mathbf{x}_j$. Here we omit the weight matrix $\Theta$ and ReLU function at the first layer for the ease of understanding.**

vital to capture the varying importance of frequencies in the filter while the fixed low-pass filters largely limit the fitting capability of GCN and its variants for learning discriminative node representations. Additionally, at deeper layers [38], the high-frequency components of input signal are too much weakened compared with the lower ones, leading to the over-smoothing problem [24].

We propose a novel adaptive frequency response filter to tackle the aforementioned problems. The developed filter is learnable and is able to adaptively adjust the varying importance of different frequencies for convolution. Toward this goal, one straightforward solution is to assign a learnable parameter for each frequency component at each layer to increase fitting capability. Nonetheless, this solution requires explicit eigendecomposition which is too expensive; meanwhile, the size of parameter space is linear w.r.t. the number of nodes, which makes the model more prone to overfitting. Our solution to address these issues is simple but elegant – we assign a single parameter per feature dimension for the low-pass filter at each layer and a more powerful filter can then be built when multiple layers are stacked together. Specifically, the filter at each layer is formulated as $\mathbf{z}_j = \mathbf{x}_j - \phi\tilde{\mathbf{L}}\mathbf{x}_j$ $(1 \le j \le F)$ for the $j$-th feature channel, where $\phi$ is a learnable parameter. We can also generalize it to a multi-channeled input signal $\mathbf{X} \in \mathbb{R}^{N\times F}$ that has $F$ different feature channels:

$$\mathbf{Z} = \mathbf{X} - \tilde{\mathbf{L}}\mathbf{X}\Phi, \tag{2}$$

where $\Phi = \text{diag}(\phi_1, \ldots, \phi_F)$, and $\phi_j$ denotes the learnable parameter for the $j$-th feature channel. We will present details on theoretical analysis of the built filter with multiple layers in Section 3.

## 2.3 Overall Architecture of AdaGNN

**Feature Representation Learning.** Based on the above discussion, we can stack $K$ layers of convolution operations as Eq. (2) for a more powerful filter. Traditionally, each layer takes the output of the previous layer as input and is transformed with a weight matrix followed by a nonlinear activation function. Inspired by [38], we can remove the noncritical nonlinear activation functions and weight matrices at intermediate layers and only keep those at the first layer. The reason is two-fold: (1) nonlinear feature transformation that may benefit learning is still preserved; (2) the number of model parameters greatly decreases by reducing the input feature channels for the second and later layers. Specifically, suppose $\mathbf{H}^{(0)} = \mathbf{X}$, then we have $\mathbf{H}^{(1)} = \text{ReLU}((\mathbf{H}^{(0)} - \tilde{\mathbf{L}}\mathbf{H}^{(0)}\Phi_1)\Theta)$ as the output of the first layer, where $\Phi_1$ (diagonal matrix) and $\Theta \in \mathbb{R}^{F\times L}$ are the learnable parameters for the filter and the weight matrix at the first layer, respectively. As mentioned before, we often set $L < F$. For the intermediate layer $2 \le k \le K$, we have $\mathbf{H}^{(k)} = \mathbf{H}^{(k-1)} - \tilde{\mathbf{L}}\mathbf{H}^{(k-1)}\Phi_k$, where $\Phi_k$ is the learnable parameters for the $k$-th layer. The output representation after $K$ layers are $\mathbf{H}^{(K)} \in \mathbb{R}^{N\times L}$. A learning illustration from the spectral perspective is shown in Fig. 1. According to the theoretical analysis in Section 3, each feature channel has its own filter, whose frequency response function can be adaptively learned to capture the useful information in different frequency components. Due to the filters for different feature channels are decoupled from each other, appropriate levels of smoothness can be individually achieved for each feature channel, which provides strengthened fitting ability.

**Label Prediction.** The output $\mathbf{H}^{(K)}$ are further fed into a softmax classifier to obtain the probability of nodes in $C$ different classes. In particular, we have $\hat{\mathbf{Y}} = \text{softmax}(\mathbf{H}^{(K)}\mathbf{W})$, where $\mathbf{W} \in \mathbb{R}^{L\times C}$ is a weight matrix to transform node representations to the label space. Then the loss function of the proposed AdaGNN framework is formulated as follows:

$$\mathcal{L} = -\sum_{i\in\mathcal{Y}_L}\sum_{j=1}^{C}\mathbf{Y}_{ij}\ln\hat{\mathbf{Y}}_{ij} + \alpha\sum_{k=1}^{K}\|\Phi_k\|_1 + \beta\Big(\sum_{k=1}^{K}\|\Phi_k\|_F^2 + \|\Theta\|_F^2 + \|\mathbf{W}\|_F^2\Big), \tag{3}$$

where $\mathcal{Y}_L$ is the set of labeled nodes indices, and $\mathbf{Y} \in \mathbb{R}^{|\mathcal{Y}_L|\times C}$ denotes the ground truth labels. The first term is the cross-entropy loss between predictions and ground truth on labeled nodes. The second term is the $\ell_1$-norm regularization of $\Phi_k$ for sparsity to preserve as much information as possible after each layer. The third term is the $\ell_2$-norm regularization for all trainable parameters to prevent overfitting. The effect of the two regularization terms are controlled by the tunable hyper-parameters $\alpha$ and $\beta$, respectively.

## 3 THEORETICAL ANALYSIS

### 3.1 Connections to GCN and GraphSAGE

**Observation 1.** *The aggregation operation of GCN reduces to the operation of AdaGNN defined in Eq. (2) when $\tilde{\mathbf{L}} = \tilde{\mathbf{L}}_{sym}$ and $\Phi = \mathbf{I}$;*

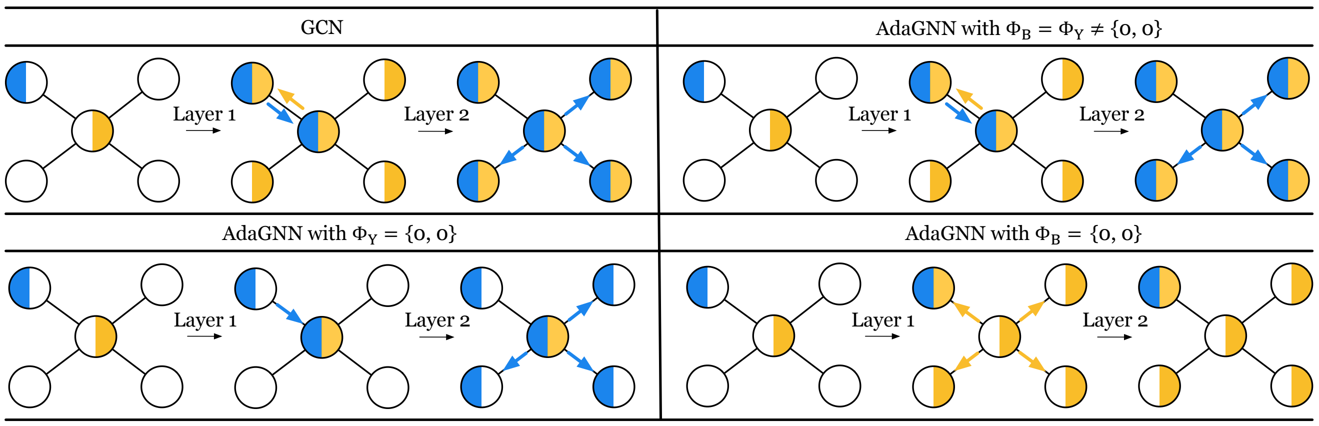

GCN
AdaGNN with $\Phi_B = \Phi_Y \neq \{0, 0\}$
AdaGNN with $\Phi_Y = \{0, 0\}$
AdaGNN with $\Phi_B = \{0, 0\}$
Layer 1
Layer 2

**Figure 2: An illustrative example to show how the learnable parameters of AdaGNN controlling the smoothness of different feature channels benefits the model fitting ability.**

*GraphSAGE aggregation operation with mean aggregator and sampling rate being 1 is also a special case when $\tilde{\mathbf{L}} = \tilde{\mathbf{L}}_{rw}$ and $\mathbf{\Phi} = \mathbf{I}$.*

The above observation reveals the inherent connections between our proposed AdaGNN and prevalent GNNs such as GCN [20] and GraphSAGE [15], and a detailed derivation can be found in the Appendix. As the aggregation operations of GCN and GraphSAGE are fundamental building blocks of modern GNN architectures [12, 42], it shows the broad generalization of our proposed framework.

## 3.2 Spectral Analysis of AdaGNN

Here we provide a formal spectral analysis of the frequency response of AdaGNN framework with $K$ layers. Spectral analysis is based on $\tilde{\mathbf{L}}_{sym}$ and here we omit the weight matrix and activation function in the first layer for ease of analysis [38].

**Theorem 1.** *For a K-layer AdaGNN framework, its frequency response function of the j-th input feature channel is formulated as $f_K(\tilde{\lambda}_i, \phi_j) = \prod_{k=1}^{K} g_k(\tilde{\lambda}_i, \phi_{j,k}) = \prod_{k=1}^{K}(1 - \phi_{j,k}\tilde{\lambda}_i)$, where $\phi_{j,k}$ denotes the learnable parameter of j-th feature channel at layer k.*

The detailed proof of the above theorem is in the Appendix. Compared with the $K$-layered GCN whose frequency response function is $f_K(\tilde{\lambda}_i) = (1 - \tilde{\lambda}_i)^K$ for any feature channel[1], the advantages of AdaGNN are three-fold: (1) the parameter $\phi_{j,k}$ can be used to adjust the relative importance of high-frequency and low-frequency components at each layer $k$; (2) when multiple layers are stacked, the importance of different frequency components can be better captured as a set of trainable parameters $\{\phi_{j,1}, ..., \phi_{j,K}\}$, yielding a more complex frequency response function; (3) the frequency response function of each feature channel is decoupled from each other, providing them with more flexibility to achieve certain levels of feature smoothness.

## 3.3 Spatial Analysis of AdaGNN

**Corollary 1.** *AdaGNN adaptively adjusts the smoothness of each feature channel via learnable parameter $\mathbf{\Phi}$ in the information aggregation process in the spatial domain.*

[1]Here we also omit the weight matrix and nonlinear transformation function at each layer for fair comparison.

Proof of Corollary 1 and comparison between AdaGNN, GCN and GraphSAGE at information aggregation process in spatial domain can be found in the Appendix. To show how AdaGNN controls the smoothness of different feature channels individually, we provide an illustrative example in Fig. 2. Here we have five nodes and two feature channels (blue and yellow). At the very beginning, the upper left node is associated with the blue channel and the middle node is with the yellow channel. For GCN, features propagate with the same mechanism and the node representations become the same after two layers. For AdaGNN, suppose the learnable parameters for these two channels across two layers are $\Phi_B = \{\phi_{B,1}, \phi_{B,2}\}$ and $\Phi_Y = \{\phi_{Y,1}, \phi_{Y,2}\}$. When the parameters for these two channels are the same, AdaGNN still suffers from the over-smoothing problem (upper right subfigure). However, by learning optimal parameter $\mathbf{\Phi}$, AdaGNN adaptively controls the smoothness of each feature channel. This example demonstrates that decoupling the smoothness of each feature channel from each other naturally makes nodes better discriminative, which alleviates the over-smoothing problem (the lower two subfigures). It should be noted that the smoothness of each feature channel can still be the same if this benefits the prediction accuracy, which implies strengthened fitting ability of our proposed model.

## 3.4 Over-smoothing Analysis

Here we show why over-smoothing is inevitable in GCN and its variants with fixed low-pass filters and why our proposed AdaGNN can naturally alleviate the over-smoothing problem.

**Theorem 2.** *For any fixed low-pass filters defined over $\tilde{\mathbf{L}}_{sym}$, we assume $\tilde{\lambda}_1$ is the smallest eigenvalue. Given a graph signal $\mathbf{x}$, suppose we convolve $\mathbf{x}$ with the filter across $K$ layers (assume the filter is $g_k(.)$ at layer $k$). If the total frequency response satisfies that $\lim_{K\to\infty} \prod_{k=1}^{K} g_k(\tilde{\lambda}_i) = 0$ $(\forall\tilde{\lambda}_i \neq \tilde{\lambda}_1)$, then over-smoothing issue is inevitable (i.e., feature values of different nodes become the same), and vice versa.*

Proof of Theorem 2 is in the Appendix. As the filter of conventional GCN and its variants are mainly defined over $\tilde{\mathbf{L}}_{sym}$ and satisfy the above condition at extremely deep layers, thus they often

**Table 1: Detailed statistics of the datasets in our experiments.**

| | BlogCatalog | ACM | Cora | Citeseer | Pubmed |
|---|---|---|---|---|---|
| **# Nodes** | 5,196 | 16,484 | 2,708 | 3,327 | 19,717 |
| **# Edges** | 173,468 | 71,980 | 5,429 | 4,732 | 44,338 |
| **# Features** | 8,189 | 8,337 | 1,433 | 3,703 | 500 |
| **# Avg. Deg.** | 66.8 | 8.7 | 4.0 | 2.8 | 4.5 |
| **# Classes** | 6 | 9 | 7 | 6 | 3 |

suffer from the over-smoothing problem. Meanwhile, we have the following corollary for AdaGNN.

**Corollary 2.** *Our AdaGNN model can naturally alleviate the over-smoothing problem at deeper layers.*

The proof of Corollary 2 is very straightforward. In Theorem 1, we have shown that for any feature channel $j$, its frequency response function over $K$ layers is $\prod_{k=1}^{K}(1-\phi_{j,k}\tilde{\lambda}_i)$, where $0 \leq \tilde{\lambda}_i < 2$ [38]. The trainable parameter $\phi_{j,k}$ can help adjust the value of frequency response to ensure it does not satisfy the condition in Theorem 2 (i.e., preventing the total frequency response from approaching 0), naturally alleviating the over-smoothing problem.

## 4 EXPERIMENTAL EVALUATIONS

In this section, we perform experiments on real-world datasets of various types to validate the effectiveness of AdaGNN. In particular, we aim to answer the following research questions – **RQ1**: How does AdaGNN perform compared with other state-of-the-art spectral GNNs and corresponding variants? **RQ2**: How well can AdaGNN alleviate the over-smoothing problems at deeper layers? **RQ3**: In which way will different frequency components contribute to learning? **RQ4**: What is the impact of the adaptive frequency response filter of the proposed AdaGNN?

### 4.1 Experimental Settings

**Datasets.** To comprehensively understand how AdaGNN works, we use five real-world attributed networks, including one social network BlogCatalog [23], one co-author network ACM [35], and three citation networks Cora, Citeseer, and Pubmed [20]. The detailed information and statistics of these datasets are shown in Table 1.

**Baselines and Evaluation Protocols.** We design two different versions of AdaGNN by instantiating $\tilde{\mathbf{L}}$ as $\tilde{\mathbf{L}}_{rw}$ and $\tilde{\mathbf{L}}_{sym}$, and we name these two implementations as AdaGNN-R and AdaGNN-S. These two methods are compared with the following state-of-the-art spectral GNNs/variants: (1) *GCN* [20]; (2) *GraphSAGE* [15]; (3) *SGC* [38]. Also, we compare our framework with other recently developed methods that tackle over-smoothing in information propagation process: (4) *DropEdge* [33] – which relieves over-smoothing issue by edge masking; (5) *PairNorm* [46] – which tackles over-smoothing with a normalization layer and its two different corresponding implementations are *PairNorm-SI* and *PairNorm-SCS*. We use the mean aggregator and assign the sampling rate of 1 for GraphSAGE. Meanwhile, *GCN* layers are used as the backbone of *DropEdge*, *PairNorm-SI*, and *PairNorm-SCS*. All methods are compared on the semi-supervised node classification task. For BlogCatalog and ACM datasets, we randomly sample 10% nodes for training, 20% for validation, and the rest 70% for test. For Cora, Citeseer, and Pubmed, we use the commonly used split as [20, 41]. Average classification accuracy on the test data is used for evaluation and all results are averaged over 10 different runs.

**Implementation Details.** The proposed AdaGNN is implemented in Pytorch [32] with Adam optimizer [19], and embedding dimensions are set to be 128 across hidden layers. ReLU and Softmax activation functions are used for the first and the last layer, and the rest layers do not use any activation functions. For the baseline methods, we use their released implementations, and the hidden unit number is also specified as 128 for a fair comparison. For BlogCatalog, Cora, Citeseer and Pubmed, we vary the number of layers in {2, 4, 8, 16} for all methods; for ACM, high feature dimension and low average node degree naturally relieve information loss in feature diffusion. Hence, over-smoothing is less obvious with above layer settings. Consequently, we vary the layer number in {2, 8, 32, 128} to have a better observation of the over-smoothing issue. Other experimental details including hyper-parameter settings and corresponding search space are also presented in the Appendix.

### 4.2 Experimental Results

In this subsection, we show the detailed experimental results w.r.t. the above research questions.

**Model Expressiveness (RQ1)** To validate the expressiveness of the proposed AdaGNN, we compare AdaGNN-R and AdaGNN-S with different baselines on semi-supervised node classification. We vary the model layer $K$ and present the performance of all models w.r.t. layer number as Fig. 3. Detailed quantitative results are in the Appendix. We make the following observations: (1) The proposed AdaGNN-R and AdaGNN-S outperform baseline methods in all cases, which demonstrates that the designed adaptive frequency response filter can indeed increase the fitting capability of the model by learning more discriminative embeddings. (2) The performance improvements of AdaGNN-R and AdaGNN-S are more obvious on BlogCatalog compared with other four datasets. The reason could be attributed to the high average node degree of social network (as indicated in Table 1) – nodes are influenced by more neighbors during neighborhood aggregation, which is consistent with the observations in previous literature [5]. For such dataset with high average node degree, the adaptive frequency response provided by AdaGNN can help achieve more appropriate feature smoothness, resulting in better performance.

**Over-smoothing (RQ2)** Now we answer RQ2 by investigating how models perform when the layer number increases. We have the following observations in Fig. 3: (1) The best performance is achieved at shallow layers for all baselines; however the performance of conventional GNNs (e.g., GCN, GraphSAGE, and SGC) drops sharply when layer goes deeper, revealing that they suffer from the over-smoothing issue at deeper layers. (2) DropEdge and PairNorms (including PairNorm-SI and -SCS) are recently proposed methods to relieve over-smoothness, whose performance does not drop as fast as conventional GNNs. In particular, the performance of DropEdge is comparable to its backbone GCN while PairNorms is inferior to its backbone in many scenarios, which is consistent with the observations in the original papers [46]. (3) The performances of the proposed AdaGNN-R and AdaGNN-S are further improved with more powerful representations at deeper layers in

**Table 2: Classification accuracy comparison between AdaGNN and different baselines. The number in the parentheses denotes the number of layers when best performance can be achieved for different methods (the performance of AdaGNN-R and AdaGNN-S is marked in bold).**

| Model | BlogCatalog | ACM | Cora | Citeseer | Pubmed |
|---|---|---|---|---|---|
| GCN | 73.98 ± 0.6% (2) | 75.55 ± 0.2% (2) | 81.30 ± 0.3% (2) | 71.30 ± 0.3% (2) | 78.58 ± 0.6% (2) |
| GraphSAGE | 70.41 ± 0.5% (2) | 75.29 ± 0.2% (2) | 80.75 ± 0.1% (2) | 70.85 ± 0.5% (2) | 78.22 ± 0.2% (2) |
| SGC | 73.97 ± 0.6% (2) | 75.14 ± 0.1% (8) | 79.86 ± 0.5% (2) | 69.13 ± 0.2% (2) | 77.60 ± 0.4% (2) |
| DropEdge | 74,17 ± 0.7% (2) | 73.05 ± 0.6% (2) | 81.20 ± 0.4% (2) | 71.20 ± 0.4% (2) | 79.33 ± 0.3% (2) |
| PairNorm-SI | 67.32 ± 0.7% (2) | 74.87 ± 0.3% (2) | 79.90 ± 0.5% (2) | 67.17 ± 0.4% (2) | 78.11 ± 0.6% (8) |
| PairNorm-SCS | 71.67 ± 0.3% (2) | 75.44 ± 0.1% (2) | 81.90 ± 0.9% (2) | 68.08 ± 1.4% (2) | 78.46 ± 0.1% (2) |
| **AdaGNN-R** | **88.32 ± 0.2% (16)** | **75.55 ± 0.1% (32)** | **82.50 ± 0.4% (8)** | **70.90 ± 0.4% (8)** | **78.90 ± 0.2% (8)** |
| **AdaGNN-S** | **89.83 ± 0.1% (4)** | **76.11 ± 0.4% (8)** | **83.60 ± 0.2% (8)** | **72.03 ± 0.1% (4)** | **79.60 ± 0.2% (8)** |

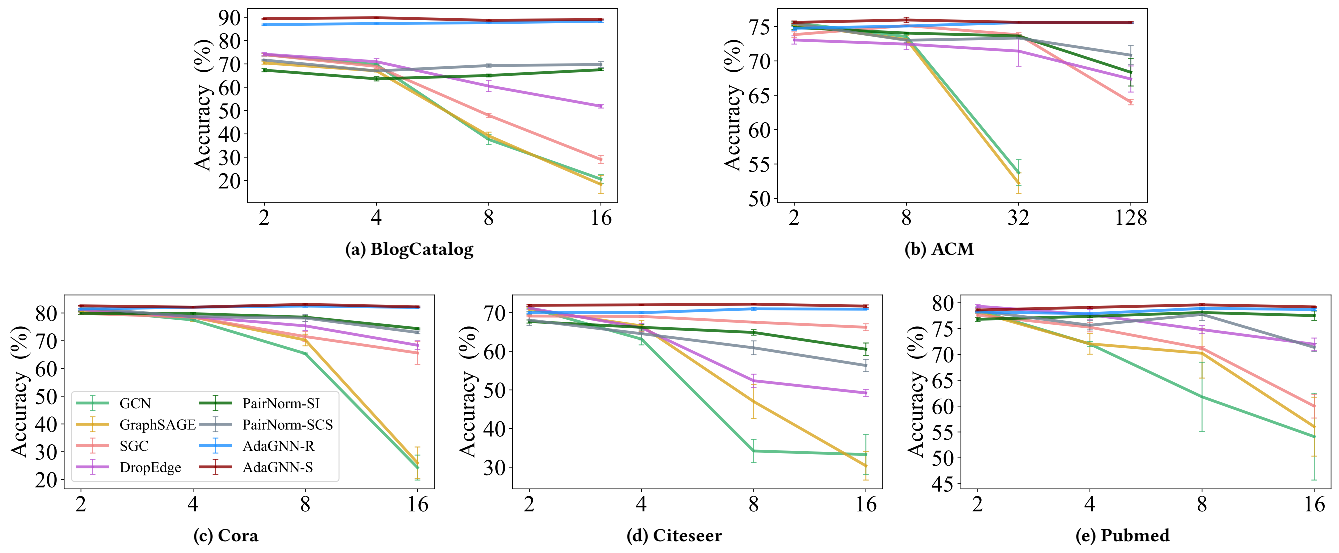


**Figure 3: Classification performance variation of different methods w.r.t. the number of model layers.**

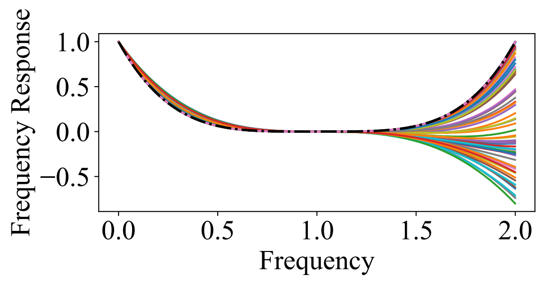
our proposed framework, demonstrating the effectiveness of the proposed filter in tackling over-smoothing and extracting more information from deeper layers.

**Filter Analysis (RQ3)** To answer RQ3, in Fig. 4a, we provide a detailed visualized comparison between the fixed filter of SGC and the learned filters of AdaGNN-S for different feature channels. It should be noted that the frequency response of the two models can be compared following the order of eigenvalues due to both AdaGNN and SGC are graph Laplacian matrix based filters [8]. Frequency response from 0 to 2 is presented without cut-off for generalization purpose. As can be shown, the learned frequency response function of AdaGNN-S varies across channels while the function of SGC treats these channels equally. Compared with the band-stop frequency response function of SGC, AdaGNN-S preserves more middle-frequency components. Besides, response of AdaGNN-S is highly selective across high-frequency components, revealing that some of them are complementary to low/middle frequencies to improve learning while some can be taken as noise.

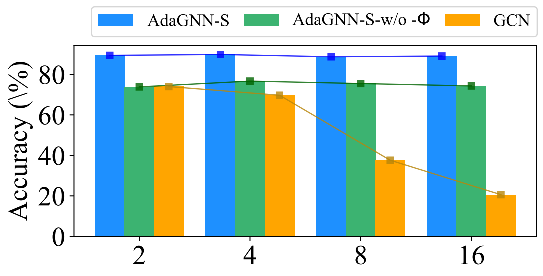
**Ablation Study (RQ4)** Now we perform ablation study to answer RQ4. AdaGNN-S reduces to GCN when we remove the parameter matrix $\mathbf{\Phi}$ and incorporate the weight matrix and activation function at each layer. Consequently, we compare the performance of three models for ablation study, i.e., AdaGNN-S, AdaGNN-S-w/o-$\mathbf{\Phi}$, and GCN. It can be observed from Fig. 4b that the learning performance is greatly reduced when $\Phi_k$ is removed at each layer. Besides, weight matrices for hidden layers make limited contribution to the learning performance as the performance of AdaGNN-S w/o $\mathbf{\Phi}$ and GCN are very similar with different layers.

## 5 RELATEX WORK

**Spectral-based Graph Neural Networks.** Existing graph neural network models are often categorized as spectral-based and spatial-based methods depending on the operation domain [39, 43]. Graph signal processing lays a solid mathematical foundation for spectral-based methods by enabling them to define graph filter in the spectral domain. Bruna et al. [4] first proposed to generalize the convolution operations in CNN to graphs and define graph



(a) Frequency response.



(b) Model ablation study.

**Figure 4: (a) Frequency response function of 4-layered AdaGNN-S and SGC on BlogCatalog. The black dashed curve denotes the response of SGC while other solid curves denote the responses of AdaGNN-S across feature channels. (b) Model ablation study of AdaGNN-S on BlogCatalog, where layer number varies from {2, 4, 8, 16}.**

filter with the spectrum of the graph Laplacian matrix. Later on, Defferrard et al. [9] proposed a fast localized convolutional filter ChebNet based on Chebyshev polynomial which avoids expensive eigendecomposition operation and is considered as a special case of CayleyNet that applies Cayley polynomials [22]. The seminal work of GCN [20] utilizes a spectral filter by truncating Chebyshev polynomial to only first order and the filter can be regarded as neighborhood aggregation in the spatial domain. SGC [38] further simplifies GCN by collapsing weight matrix in consecutive layers and showed that there are redundant computations in GCN. Additionally, recent research efforts attempt to improve the spectral filter from different perspectives, such as learning hidden structural relations [25] and capturing both local and global information [49]. Despite their solid mathematical foundation and empirical effectiveness, these attempts, however, cannot well characterize the varying importance of frequencies for learning.

**Spatial-based Graph Neural Networks.** Spatial-based methods perform convolution in the spatial domain by aggregating and transforming the information of neighboring nodes. Different methods in this family mainly differ in the way how the aggregation function is designed. One of the earliest attempts NN4G [28] sums up the information from a central node's neighbors. DCNN [1] regards graph convolutions as a diffusion process w.r.t. specific probabilities to attain equilibrium after several rounds. To better distinguish the importance of different neighbors for information aggregation, attention mechanism is also utilized in GAT [36]. MPNN [13] generalizes different spatial-based methods as a unified message-passing framework. GraphSAGE [15] aggregates neighborhood information via mean/max/LSTM pooling. GIN [40] allocates a learnable parameter for the center node when performing information aggregation, which empowers the model stronger capability to differentiate different graph structures. More recently, a myriad of more sophisticated aggregation strategies compared with previous works are developed and a more detailed review can be referred to [39, 45].

**Over-smoothing of Graph Neural Networks.** Information aggregation from node neighbors is a critical step of GNNs, which smoothes node feature representations over the whole graph [2, 47, 48]. In the spectral domain, this can be interpreted as weakening the high-frequency components of input signals. For example, studies have shown that the convolution operation in GCN corresponds to a single fixed low-pass filter [30]. When multiple convolution layers are stacked and the model goes deeper, the over-smoothing issue is inevitable, which means node representations converge to similar values. Li et al. [24] proved that GCN is actually a kind of Laplacian smoothing process, and proposed the challenge of over-smoothing for the first time. After that, some studies demonstrate that certain level of smoothness benefits node representation learning while over-smoothing broadly exists in deeper GNNs [5, 10]. More recently, some researches attempt to relieve this problem via residual-like connections [6, 26, 27]. Nevertheless, such methods are unable to avoid the situation where a node is overwhelmed in the information of its neighbors in the information aggregation process. There are also works directly relieve over-smoothing in this process via using either edge masking [33] or re-normalization [46]; however performance still obviously reduces when models go deeper. Consequently, it remains a challenging problem to directly relieve over-smoothing in the information propagating process. To the best of our knowledge, we are the first to provide an understanding of this problem from the perspective of the spectral filter, and our experiments also demonstrate its superiority over other prevalent solutions such as DropEdge [33] and PairNorm [46].

## 6 CONCLUSION

Existing GNNs mainly apply fixed filters for the convolution operation, thus their expressiveness is limited and suffers from the over-smoothing problem. To tackle the above problems, in this paper, we propose a novel framework AdaGNN with an adaptive frequency response filter. The proposed filter is able to adaptively adjust the importance of different frequency components of each input feature channel and achieve different levels of smoothness for different feature channels. We provide theoretical analysis for the proposed AdaGNN from different aspects, and empirical experimental evaluations also demonstrate its superiority over state-of-the-art GNNs. We will leave the fairness issues of the proposed AdaGNN framework as our future research directions.

## 7 ACKNOWLEDGEMENTS

This material is, in part, supported by the National Science Foundation (NSF) under grants #2006844.

## 8 APPENDIX

## A PROOF OF OBSERVATION 1.

Proof. The information aggregation of AdaGNN with $\tilde{\mathbf{L}}_{sym}$ and $\tilde{\mathbf{L}}_{rw}$ at layer $k$ can be respectively formulated as follows:

$$\mathbf{E}_{v,j}^{(k)} = \mathbf{H}_{v,j}^{(k)} - \phi_{j,k} \sum_{u \in \mathcal{N}(v)} (\tilde{\mathbf{L}}_{sym})_{v,u} \mathbf{H}_{u,j}^{(k)}, \quad \text{and} \tag{4}$$

$$\mathbf{E}_{v,j}^{(k)} = \mathbf{H}_{v,j}^{(k)} - \phi_{j,k} \sum_{u \in \mathcal{N}(v)} (\tilde{\mathbf{L}}_{rw})_{v,u} \mathbf{H}_{u,j}^{(k)}, \tag{5}$$

Considering that

$$(\tilde{\mathbf{L}}_{sym})_{v,u} = \begin{cases} -\frac{1}{\sqrt{|\mathcal{N}(v)|+1}\sqrt{|\mathcal{N}(u)|+1}}, & \text{if } u \in \mathcal{N}(v), \\ \frac{|\mathcal{N}(v)|}{|\mathcal{N}(v)|+1}, & \text{if } u = v, \\ 0, & \text{otherwise,} \end{cases} \quad \text{and}$$

$$(\tilde{\mathbf{L}}_{rw})_{v,u} = \begin{cases} -\frac{1}{|\mathcal{N}(v)|+1}, & \text{if } u \in \mathcal{N}(v), \\ \frac{|\mathcal{N}(v)|}{|\mathcal{N}(v)|+1}, & \text{if } u = v, \\ 0, & \text{otherwise,} \end{cases}$$

we have the following two formulations:

$$\mathbf{E}_{v,j}^{(k)} = (1 - (1 - \frac{1}{|\mathcal{N}(v)|+1})\phi_{j,k})\mathbf{H}_{v,j}^{(k)} + \phi_{j,k} \sum_{u \in \mathcal{N}(v)} \frac{\mathbf{H}_{u,j}^{(k)}}{\sqrt{|\mathcal{N}(v)|+1}\sqrt{|\mathcal{N}(u)|+1}}, \quad \text{and} \tag{6}$$

$$\mathbf{E}_{v,j}^{(k)} = (1 - (1 - \frac{1}{|\mathcal{N}(v)|+1})\phi_{j,k})\mathbf{H}_{v,j}^{(k)} + \phi_{j,k} \sum_{u \in \mathcal{N}(v)} \frac{\mathbf{H}_{u,j}^{(k)}}{|\mathcal{N}(v)|+1}. \tag{7}$$

Then if we replace all $\phi_{j,k}$ by 1, and Eq. (6) and Eq. (7) can be respectively reformulated as

$$\mathbf{E}_{v,j}^{(k)} = \frac{1}{|\mathcal{N}(v)|+1}\mathbf{H}_{v,j}^{(k)} + \sum_{u \in \mathcal{N}(v)} \frac{\mathbf{H}_{u,j}^{(k)}}{\sqrt{|\mathcal{N}(v)|+1}\sqrt{|\mathcal{N}(u)|+1}}, \quad \text{and} \tag{8}$$

$$\mathbf{E}_{v,j}^{(k)} = \frac{1}{|\mathcal{N}(v)|+1}\mathbf{H}_{v,j}^{(k)} + \sum_{u \in \mathcal{N}(v)} \frac{\mathbf{H}_{u,j}^{(k)}}{|\mathcal{N}(v)|+1}. \tag{9}$$

Therefore, the convolution operation of AdaGNN with $\tilde{\mathbf{L}}_{sym}$ and $\mathbf{\Phi} = \mathbf{I}$ equals to that of GCN. Also, the convolution operation of AdaGNN with $\tilde{\mathbf{L}}_{rw}$ and $\mathbf{\Phi} = \mathbf{I}$ equals to that of GraphSAGE with mean aggregator and sampling rate being 1. □

**Table 3: Hyper-parameters search space for different baselines.**

| Model | Epoch ($\times 10^2$) | Learning Rate | L2 Regularization | Dropout |
|---|---|---|---|---|
| GCN | {2, 3, 5, 8} | {1e-2, 1e-3, 1e-4} | {5e-4, 9e-4, 9e-8} | {0.1, 0.2, 0.3, 0.4, 0.5} |
| GraphSAGE | {2, 3, 5, 8} | {1e-2, 1e-3, 1e-4} | {5e-4, 9e-4, 9e-8} | {0.1, 0.2, 0.3, 0.4, 0.5} |
| SGC | {2, 3, 5, 8} | {1e-2, 1e-3, 1e-4} | {5e-4, 9e-4, 9e-8} | {0.1, 0.2, 0.3, 0.4, 0.5} |
| DropEdge | {5, 8, 15, 20} | {2e-2, 1e-2, 1e-3} | {5e-4, 9e-4, 9e-8} | {0.1, 0.2, 0.3, 0.4, 0.5} |
| PairNorm | {8, 10, 15, 20} | {1e-2, 5e-3, 1e-3} | {5e-4, 9e-4, 9e-8} | {0.1, 0.2, 0.3, 0.4, 0.5} |

## B PROOF OF THEOREM 1.

Proof. As mentioned in Section 3.2, we replace $\tilde{\mathbf{L}}$ with $\tilde{\mathbf{L}}_{sym}$ and omit the weight matrix and activation function for the frequency response function proof. Consider the $j$-th feature channel $\mathbf{x}_j \in \mathbb{R}^N$ of the input signal at the $k$-th layer, then we have

$$\begin{aligned} \mathbf{x}_j - \phi_{j,k}\tilde{\mathbf{L}}_{sym}\mathbf{x}_j &= \mathbf{x}_j - \phi_{j,k}\tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^T\mathbf{x}_j \\ &= (\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T - \phi_{j,k}\tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^T)\mathbf{x}_j \\ &= \tilde{\mathbf{U}}(\mathbf{I} - \phi_{j,k}\tilde{\mathbf{\Lambda}})\tilde{\mathbf{U}}^T\mathbf{x}_j. \end{aligned}$$

As a consequence, the frequency response of AdaGNN can be derived as $g_k(\tilde{\lambda}_i) = 1 - \phi_{j,k}\tilde{\lambda}_i$. If we stack $K$ layers together, then the overall frequency response function can be derived as $f_K(\tilde{\lambda}_i, \phi_j) = \prod_{k=1}^{K} g_k(\tilde{\lambda}_i, \phi_{j,k}) = \prod_{k=1}^{K}(1 - \phi_{j,k}\tilde{\lambda}_i)$. □

## C PROOF OF THEOREM 2.

Proof. Consider a graph signal $\mathbf{x} \in \mathbb{R}^N$ as input. Here we assume that the value of at least one dimension in $\mathbf{x}$ is different from other dimensions, i.e., $\mathbf{x}$ is not an over-smoothed signal and $N \geq 2$.

**Fact 1.** *For any $\tilde{\mathbf{L}}_{sym} = \tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^T \in \mathbb{R}^{N \times N}$ of an undirected graph, columns in $\tilde{\mathbf{U}}$ are orthogonal to each other, and eigenvector $\tilde{\mathbf{u}}_1$ corresponding to the smallest eigenvalue $\tilde{\lambda}_1$ is collinear with the vector* $[1, 1, 1, ..., 1]$ *of length $N$.*

As such, we can regard $\tilde{\mathbf{L}}_{sym}\mathbf{x} = \tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^T\mathbf{x}$ as a process of projecting $\mathbf{x}$ onto $N$ eigenvectors, then re-weighting the length of each component vector and summing them together. Assume weight of each frequency component (i.e., each $\tilde{\lambda}_i$, $1 \leq i \leq N$) of the filter defined over $\tilde{\mathbf{L}}_{sym}$ at filtering time $k$ ($1 \leq k \leq K$) being $g_k(\tilde{\lambda}_i)$, then $\mathbf{x}_K$, i.e., $\mathbf{x}$ being filtered $K$ times, will be

$$\mathbf{x}_K = \prod_{k=1}^{K} g_k(\tilde{\lambda}_1)\frac{\mathbf{u}_1 \cdot \mathbf{x}}{|\mathbf{u}_1|^2}\mathbf{u}_1 + \prod_{k=1}^{K} g_k(\tilde{\lambda}_2)\frac{\mathbf{u}_2 \cdot \mathbf{x}}{|\mathbf{u}_2|^2}\mathbf{u}_2 + ... + \prod_{k=1}^{K} g_k(\tilde{\lambda}_N)\frac{\mathbf{u}_N \cdot \mathbf{x}}{|\mathbf{u}_N|^2}\mathbf{u}_N. \tag{10}$$

**Fact 2.** *If all entries of an $N$-dimensional nonzero vector are the same, then this vector is collinear with the vector* $[1, 1, 1, ..., 1]$ *of length $N$ ($N \geq 2$).*

As such, $\mathbf{x}_K$ is collinear with $\mathbf{u}_1$ when $K \longrightarrow \infty$ (i.e., over-smoothing issue being inevitble when $K$ gets larger) iff $\lim_{K \to \infty} \prod_{k=1}^{K} g_k(\tilde{\lambda}_i) = 0$ $(\forall \tilde{\lambda}_i \neq \tilde{\lambda}_1)$. □

## D HYPER-PARAMETER SETTINGS.

For two layered AdaGNN-R and AdaGNN-S, we fix the learning rate as 0.01 with exponential decay step length tuned between 30 and 50. Total training epochs are tuned between 80 and 300. $\alpha$ is fixed as 1e-6 and $\beta$ is tuned between 3e-4 and 9e-4. Dropout rate is tuned between 0.5 and 0.8. The parameters of AdaGNN-R and AdaGNN-S with deeper layers are initialized with the learned parameters of the corresponding two-layered models. For deeper layers, the learning rate is tuned between 3e-4 and 1e-3. Total training epochs are tuned between 50 and 100. $\alpha$ is tuned between 1e-4 and 1e-6, and $\beta$ is tuned between 9e-12 and 9e-6. Dropout rate is tuned between 0.1 and 0.6. Also, a detailed hyper-parameter search space of different baselines is presented in Table 3.

**Observation 2.** *The aggregation of GCN and AdaGNN on $\tilde{\mathbf{L}}_{sym}$ can be respectively formulated as follows:*

$$\mathbf{E}_{v,j}^{(k)} = \frac{1}{|\mathcal{N}(v)|+1}\mathbf{H}_{v,j}^{(k)} + \sum_{u\in\mathcal{N}(v)} \frac{\mathbf{H}_{u,j}^{(k)}}{\sqrt{|\mathcal{N}(v)|+1}\sqrt{|\mathcal{N}(u)|+1}}, \text{ and} \tag{11}$$

$$\begin{aligned}\mathbf{E}_{v,j}^{(k)} =& (1-(1-\frac{1}{|\mathcal{N}(v)|+1})\phi_{j,k})\mathbf{H}_{v,j}^{(k)} \\ &+ \phi_{j,k}\sum_{u\in\mathcal{N}(v)} \frac{\mathbf{H}_{u,j}^{(k)}}{\sqrt{|\mathcal{N}(v)|+1}\sqrt{|\mathcal{N}(u)|+1}}.\end{aligned} \tag{12}$$

*Meanwhile, the aggregation of GraphSAGE with mean aggregator and AdaGNN on $\tilde{\mathbf{L}}_{rw}$ can be respectively formulated as follows:*

$$\mathbf{E}_{v,j}^{(k)} = \frac{1}{|\mathcal{N}(v)|+1}\mathbf{H}_{m,j}^{(k)} + \sum_{u\in\mathcal{N}(v)} \frac{\mathbf{H}_{u,j}^{(k)}}{|\mathcal{N}(v)|+1}, \text{ and} \tag{13}$$

$$\mathbf{E}_{v,j}^{(k)} = (1-(1-\frac{1}{|\mathcal{N}(v)|+1})\phi_{j,k})\mathbf{H}_{v,j}^{(k)} + \phi_{j,k}\sum_{u\in\mathcal{N}(v)} \frac{\mathbf{H}_{u,j}^{(k)}}{|\mathcal{N}(v)|+1}. \tag{14}$$

In the above formulations, $\mathbf{H}_{i,j}^{(k)}$ and $\mathbf{E}_{i,j}^{(k)}$ denote the value of the $j$-th feature on the $i$-th node before and after aggregation at layer $k$, respectively. Compared with GCN and GraphSAGE in which different feature channels have the same aggregating mechanism, in AdaGNN the parameter $\phi_{j,k}$ for each feature channel allows it to learn appropriate aggregation mechanism for a certain level of smoothness.

**Implementation Details.** The proposed AdaGNN is implemented in Pytorch [32] with Adam optimizer [19], and the embedding dimensions are set to be 128 across all layers except the first layer and the last layer. ReLU and Softmax activation functions are used for the first and the last layer, and the rest layers do not use any activation functions. For the baseline methods, we use their released implementations, and the hidden unit number is also specified as 128 for a fair comparison. For BlogCatalog, Cora, Citeseer and Pubmed, we vary the number of layers in {2, 4, 8, 16} for all methods; for ACM, high feature dimension and low average node degree naturally relieve information loss in feature diffusion. Hence, over-smoothing is less obvious with above layer settings. Consequently, we vary the layer number in {2, 8, 32, 128} to have a better observation of the over-smoothing issue. Early stopping is used for model training. To train models with 2 layers, the maximum number of epochs is 300, learning rate is 0.01, dropout rate is 0.5, $\alpha$ of $\ell_1$-norm (only for AdaGNN) is 1e-6, $\beta$ of $\ell_2$-norm (for all methods) is 9e-4. For models with deeper layers, these hyper-parameters (e.g., learning rate, dropout rate) are selected according to the best performance on the validation set, and a detailed search space of hyper-parameters is in the Appendix.

## REFERENCES

[1] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1993–2001.

[2] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research* 7, Nov (2006), 2399–2434.

[3] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. 2021. Beyond Low-frequency Information in Graph Convolutional Networks. *arXiv preprint arXiv:2101.00797* (2021).

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).

[5] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2019. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. *arXiv preprint arXiv:1909.03211* (2019).

[6] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*. PMLR, 1725–1735.

[7] Yunpeng Chen, Haoqi Fan, Bing Xu, Zhicheng Yan, Yannis Kalantidis, Marcus Rohrbach, Shuicheng Yan, and Jiashi Feng. 2019. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. In *Proceedings of 2019 IEEE International Conference on Computer Vision*. 3435–3444.

[8] Fan RK Chung and Fan Chung Graham. 1997. *Spectral graph theory*. Number 92. American Mathematical Soc.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.

[10] Zhijie Deng, Yinpeng Dong, and Jun Zhu. 2019. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192* (2019).

[11] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *Proceedings of the 2019 World Wide Web Conference*. 417–426.

[12] Hongyang Gao and Shuiwang Ji. 2019. Graph u-nets. *arXiv preprint arXiv:1905.05178* (2019).

[13] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*. JMLR. org, 1263–1272.

[14] Rafael C Gonzales and Richard E Woods. 2002. Digital image processing.

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[16] Kaiming He, Jian Sun, and Xiaoou Tang. 2010. Guided image filtering. In *Proceedings of 2010 European Conference on Computer Vision*. 1–14.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[18] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. *arXiv preprint arXiv:1408.5882* (2014).

[19] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[20] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based Learning Applied to Document Recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[22] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. 2018. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing* 67, 1 (2018), 97–109.

[23] Jundong Li, Xia Hu, Jiliang Tang, and Huan Liu. 2015. Unsupervised streaming feature selection in social media. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. 1041–1050.

[24] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

[25] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive graph convolutional neural networks. In *Proceedings of the 32nd AAAI conference on Artificial Intelligence*.

[26] Wei Li, Shuheng Li, Shuming Ma, Yancheng He, Deli Chen, and Xu Sun. 2019. Recursive Graphical Neural Networks for Text Classification. *arXiv preprint arXiv:1909.08166* (2019).

[27] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 338–348.

[28] Alessio Micheli. 2009. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks* 20, 3 (2009), 498–511.

[29] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *Proceedings of 2016 International Conference on Machine Learning*. 2014–2023.

[30] Hoang NT and Takanori Maehara. 2019. Revisiting Graph Neural Networks: All We Have is Low-Pass Filters. *arXiv preprint arXiv:1905.09550* (2019).

[31] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2018. Adversarially regularized graph autoencoder for graph embedding. *arXiv preprint arXiv:1802.04407* (2018).

[32] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).

[33] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *Proceedings of 2019 International Conference on Learning Representations*.

[34] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* 30, 3 (2013), 83–98.

[35] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 990–998.

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[37] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[38] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. 2019. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153* (2019).

[39] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[40] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[41] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861* (2016).

[42] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*. 4800–4810.

[43] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. 2019. Graph convolutional networks: a comprehensive review. *Computational Social Networks* 6, 1 (2019), 11.

[44] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *Advances in Neural Information Processing Systems*. 649–657.

[45] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[46] Lingxiao Zhao and Leman Akoglu. 2019. PairNorm: Tackling Oversmoothing in GNNs. *arXiv preprint arXiv:1909.12223* (2019).

[47] Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *Advances in Neural Information Processing Systems*. 321–328.

[48] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning*. 912–919.

[49] Chenyi Zhuang and Qiang Ma. 2018. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*. 499–508.

[50] Dongmian Zou and Gilad Lerman. 2019. Encoding robust representation for graph generation. In *Proceedings of 2019 International Joint Conference on Neural Networks*. IEEE, 1–9.