Adaptive Kernel Graph Neural Network

Mingxuan Ju[1,2], Shifu Hou[2], Yujie Fan[2], Jianan Zhao[1,2], Yanfang Ye[1,2*], Liang Zhao[3]

[1] University of Notre Dame, Notre Dame, IN 46556
[2] Case Western Reserve University, Cleveland, OH 44106
[3] Emory University, Atlanta, GA 30322
{mju2, jzhao8, yye7}@nd.edu, {sxh1055,yxf370}@case.edu, liang.zhao@emory.edu

Abstract

Graph neural networks (GNNs) have demonstrated great success in representation learning for graph-structured data. The layer-wise graph convolution in GNNs is shown to be powerful at capturing graph topology. During this process, GNNs are usually guided by pre-defined kernels such as Laplacian matrix, adjacency matrix, or their variants. However, the adoptions of pre-defined kernels may restrain the generalities to different graphs: mismatch between graph and kernel would entail sub-optimal performance. For example, GNNs that focus on low-frequency information may not achieve satisfactory performance when high-frequency information is significant for the graphs, and vice versa. To solve this problem, in this paper, we propose a novel framework - i.e., namely Adaptive Kernel Graph Neural Network (AKGNN) - which learns to adapt to the optimal graph kernel in a unified manner at the first attempt. In the proposed AKGNN, we first design a data-driven graph kernel learning mechanism, which adaptively modulates the balance between all-pass and low-pass filters by modifying the maximal eigenvalue of the graph Laplacian. Through this process, AKGNN learns the optimal threshold between high and low frequency signals to relieve the generality problem. Later, we further reduce the number of parameters by a parameterization trick and enhance the expressive power by a global readout function. Extensive experiments are conducted on acknowledged benchmark datasets and promising results demonstrate the outstanding performance of our proposed AKGNN by comparison with state-of-the-art GNNs. The source code is publicly available at: https://github.com/jumxglhf/AKGNN.

Introduction

Graph-structured data have become ubiquitous in the real world, such as social networks, knowledge graphs, and molecule structures. Mining and learning expressive node representations on these graphs can contribute to a variety of realistic challenges and applications. The emphasis of this work lies in the node representation learning on graphs, aiming to generate node embeddings that are expressive with respect to downstream tasks such as node classification (Kipf and Welling 2016; Klicpera, Bojchevski, and Günnemann 2019; Veličković et al. 2017). Current state-of-the-art frameworks could be categorized as graph convolutions, where nodes aggregate information from their direct neighbors with fixed guiding kernels, e.g., different versions of adjacency or Laplacian matrices. And information of high-order neighbors could be captured in an iterative manner by stacked convolution layers.

Although the results are promising, recent researches have shown that such a propagation mechanism entails certain challenges. Firstly, (Chen et al. 2020a; Oono and Suzuki 2019) illustrate that the performance of GNNs could be deteriorated by over-smoothing if excessive layers are stacked. As proved in the work of (Zhu et al. 2021), graph convolution could be summarized as a case of Laplacian smoothing, in which adjacent nodes become inseparable after multiple layers. (Oono and Suzuki 2019) shows that multiple non-linearity functions between stacked convolution layers further antagonize this problem. Moreover, these aforementioned propagation layers cause the over-fitting problem (Wang et al. 2019). In current GNNs, each layer serves as a parameterized filter where graph signals are first amplified or diminished and then combined. Adding more layers aims to capture high-order information beneficial for the downstream classification but it meanwhile introduces the number of trainable parameters, which might cancel out the intended benefits as real-world data is often scarcely labeled (Zhao et al. 2020; Chen et al. 2020a). Effective frameworks such as JK-Nets (Xu et al. 2018b) and DAGNN (Liu, Gao, and Ji 2020) overcome the over-smoothing issue by a global readout function between propagation layers, making local information from early layers directly accessible during the inference phase. And the over-fitting issue is conquered by a single learnable matrix, placed before all propagation layers, to approximate parameters of all layers (Wu et al. 2019).

Another far less researched issue rests in the fixed graph kernels (e.g., adjacency matrix, Laplacian matrix, or their variants) that current GNNs model on, which restricts their generalization to different graphs. (Ma et al. 2020; Zhu et al. 2021; Dong et al. 2016) prove that the current GNNs could be explained in a unified framework, where the output node representation minimizes two terms: 1) the distances between adjacent nodes and 2) the similarities between the input and output signals. Some GNNs such as GCN (Kipf and Welling 2016) mainly focus on the lat-

*Corresponding Author.

ter term, which solely extracts low-frequency information. Others like APPNP (Klicpera, Bojchevski, and Günnemann 2019) merges these two terms by introducing original signals through teleport connection after low-pass filtering, and hence brings a certain degree of high-frequency signals. But in reality, it is difficult to determine what and how much information should be encoded, unless experiments are conducted across algorithms with different hyperparameters. Merely considering one kind of information might not satisfy the needs of various downstream tasks, while introducing extra information could jeopardize the decision boundary. Some very recent works such as GNN-LF and GNN-HF (Zhu et al. 2021) utilize models with different pre-defined graph kernels to adapt to various datasets. These models either focus on high or low frequency signals. Still, experiments need to be conducted on both in order to know which one works out best, and the threshold between low and high frequency signals needs to be defined via human experts, which might be sub-optimal under some circumstances. To the best of our knowledge, there has not yet a unified framework that solves this problem. To fill this research gap, in this paper, we propose Adaptive Kernel Graph Neural Network (i.e., AKGNN) where a novel adaptive kernel mechanism is devised to self-learn the optimal threshold between high and low frequency signals for downstream tasks.

Specifically, to effectively combat the generality issue entailed by fixed graph kernel, AKGNN dynamically adjusts the maximal eigenvalue of graph Laplacian matrix at each layer such that the balance between all-pass and low-pass filters is dynamically optimized. And through this process, the optimal trade-off between high and low frequency signals is learned. From the spatial point of view, our model is able to raise the weight of self-loop when neighbors are not informative (i.e., all-pass filter from spectral view), or focus more on adjacent nodes when neighbors are helpful (i.e., low-pass filter). To prevent the over-fitting problem, we modify the parameterization trick proposed in (Wu et al. 2019), wherein learnable parameters of all convolution layers, except maximal eigenvalues, are compressed and approximated by a single matrix. Nevertheless, it is possible that different nodes require information from neighbors of distinct orders and hence we utilize a global readout function (Xu et al. 2018b) to capture node embeddings at different orders.

Finally, we demonstrate the legitimacy of the proposed AKGNN through theoretical analysis and empirical studies, where it is able to achieve state-of-the-art results on node classification at acknowledged graph node classification benchmark datasets, and persistently retain outstanding performance even with an exaggerated amount of convolution layers across all benchmark datasets.

Problem and Related Work

Let $G = (V, E)$ denote a graph, in which V is the set of $|V| = N$ nodes and $E \subseteq V \times V$ is the set of $|E|$ edges between nodes. Adjacency matrix is denoted as $\mathbf{A} \subseteq \{0, 1\}^{N \times N}$. The element a_{ij} in i-th row and j-th column of $\mathbf{A}$ equals to 1 if there exists an edge between nodes v_i and v_j or equals to 0 otherwise. Laplacian matrix of a graph is denoted as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ or its normalized form $\mathbf{L} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{D}$ is the diagonal degree matrix $\mathbf{D} = diag(d(v1), \ldots, d(v_N))$ and $\mathbf{I}$ is the identity matrix. Spectral graph theories have studied the properties of a graph by analyzing the eigenvalues and eigenvectors of $\mathbf{L}$ (Kipf and Welling 2016; Defferrard, Bresson, and Vandergheynst 2016), and our model adaptively modifies the maximal eigenvalue of $\mathbf{L}$ to learn the optimal trade-off between all-pass and low-pass filters.

Node Classification on Graphs. We focus on node classification on graph. $\mathbf{X} \in \mathbb{R}^{N \times d}$ represents the feature matrix of a graph, where node v_i is given with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$ and d is the dimension size. $\mathbf{Y} \subseteq \{0, 1\}^{N \times C}$ denotes the label matrix of a graph, where C is the number of total classes. Given M labeled nodes ($0 < M \ll N$) with label $\mathbf{Y}^L$ and $N - M$ unlabeled nodes with missing label $\mathbf{Y}^U$, the objective of node classification is learning a function $f : G, \mathbf{X}, \mathbf{Y}^L \rightarrow \mathbf{Y}^U$ to predict missing labels $\mathbf{Y}^U$. Traditional solutions to this problem are mainly based on Deepwalk (Ando and Zhang 2007; Pang and Cheung 2017; Dong et al. 2016). Recently GNNs have emerged as a class of powerful approaches for this problem. GCN (Kipf and Welling 2016), which iteratively approximates Chebyshev polynomials proposed by (Defferrard, Bresson, and Vandergheynst 2016), has motivated numerous novel designs. Some typical GNNs are reviewed below.

Graph Neural Networks. GNNs generalize neural network into graph-structured data (Scarselli et al. 2008; Kipf and Welling 2016; Klicpera, Bojchevski, and Günnemann 2019; Veličković et al. 2017). The key operation is graph convolution, where information is routed from each node its neighbors with some deterministic rules (e.g., adjacency matrices and Laplacian matrices). For example, the propagation rule of GCN (Kipf and Welling 2016) could be formulated as $\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)})$, where $\hat{\mathbf{A}}$ denotes the normalized adjacency matrix with self-loop, $\sigma(.)$ denotes the non-linearity function, and $\mathbf{W}^{(l)}$ and $\mathbf{H}^{(l)}$is the learnable parameters and node representations at l^{th} layer respectively. That of APPNP (Klicpera, Bojchevski, and Günnemann 2019) is formulated as $\mathbf{Z}^{(k+1)} = (1 - \alpha)\hat{\mathbf{A}}\mathbf{Z}^{(k)} + \alpha\mathbf{H}$, where α denotes the teleport probabilities, $\mathbf{H}$ is the predicted class distribution before propagation and $\mathbf{Z}^{(k)}$ denotes the propagated class distribution at k^{th} layer.

From Graph Spectral Filter to Graph Neural Network. The idea behind graph spectral filtering is modulating the graph signals with learnable parameters so that signals at different frequencies are either amplified or diminished. (Defferrard, Bresson, and Vandergheynst 2016) proposes signal modulating with Chebyshev polynomials, which allows the model to learn neighbor information within K-hops with a relatively scalable amount of parameters. GCN proposed by (Kipf and Welling 2016) approximates the K-th order Chebyshev polynomials by K convolution layers connected back-to-back, each of which assumes K equals to 1 and the maximal eigenvalue of Laplacian matrix equals to 2. Spatially, each propagation layer could be understood as gathering information from direct neighbors by mean pooling, and information from high-order neighbors could be captured through multiple stacked layers. Our proposed AKGNN simplifies this complexity of such filtering process by decoupling it into two sub-tasks: 1) limiting the scope of filters by finding the optimal trade-off between high and low frequency signals and 2) graph signal filtering on the distilled signals.

Methodology

In this section, we explain the technical details of Adaptive Kernel Graph Neural Network (AKGNN), as shown in Fig. 1. We present a type of graph convolution that is able to adaptively tune the weights of all-pass and low-pass filters by learning the maximal eigenvalue of graph Laplacian at each layer. Through such a design, the threshold between high and low frequency signals is efficiently optimized. Demonstrated by comprehensive experiments, proposed AKGNN is able to achieve state-of-the-art results on benchmarks acknowledged as community conventions.

Adaptive Graph Kernel Learning

Given an input graph G and its normalized Laplacian matrix $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, where $\mathbf{U}$ is the eigenvector matrix and $\mathbf{\Lambda}$ is the diagonal eigenvalue matrix of $\mathbf{L}$, Cheby-Filter (Defferrard, Bresson, and Vandergheynst 2016) on a graph signal $\mathbf{f}$ is formulated as:

$$\mathbf{f}^{'} = \sum_{k=0}^{K} \theta_k \mathbf{U} T_k(\tilde{\mathbf{\Lambda}})\mathbf{U}^T\mathbf{f} = \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{f}, \quad (1)$$

where $\mathbf{f}^{'}$ is the resulted modulated signal, K is the order of the truncated polynomials, θ_k denotes the learnable filter at k^{th} order, $T_k(.)$ refers to the k^{th} polynomial bases, $\tilde{\mathbf{\Lambda}}$ denotes the normalized diagonal eigenvalue matrix, and $\tilde{\mathbf{L}} = \mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{U}^T$. $\tilde{\mathbf{\Lambda}} = \frac{2\cdot\mathbf{\Lambda}}{\lambda_{max}} - \mathbf{I}$, where λ_{max} denotes the maximum value in $\mathbf{\Lambda}$, has domain in range [-1,1]. Normalized form is used here instead of $\mathbf{\Lambda}$ since Chebyshev polynomial is orthogonal only in the range [-1,1] (Defferrard, Bresson, and Vandergheynst 2016).

GCN (Kipf and Welling 2016) simplifies the Cheby-Filter by assuming $K = 1$ and $\lambda_{max} \approx 2$ at each layer. Although the efficacy of GCN is promising, through this simplification, one issue is brought: graph convolution operation essentially conducts low-frequency filtering as proved by (Zhu et al. 2021), where the similarity between adjacent nodes are enlarged as the number of propagation layers increases, and the kernel could not be adjusted to dataset where high-frequency information is important. Given that $T_0(\tilde{\mathbf{L}}) = \mathbf{I}$ and $T_1(\tilde{\mathbf{L}}) = \tilde{\mathbf{L}}$ (Defferrard, Bresson, and Vandergheynst 2016), Eq. 1 is re-formulated as follows:

$$\begin{aligned}\mathbf{f}^{'} &= \theta_0\mathbf{I}\mathbf{f} + \theta_1(\frac{2}{\lambda_{max}}(\mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}) - \mathbf{I})\mathbf{f} \\ &= \theta_0\mathbf{I}\mathbf{f} + \frac{2}{\lambda_{max}}\theta_1\mathbf{I}\mathbf{f} - \frac{2}{\lambda_{max}}\theta_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{f} - \theta_1\mathbf{I}\mathbf{f}.\end{aligned} \quad (2)$$

By setting $\theta_0 = -\theta_1$, Eq. 2 can be simplified as follows:

$$\begin{aligned}\mathbf{f}^{'} &= \theta_0\mathbf{I}\mathbf{f} + \frac{2}{\lambda_{max}}\theta_1\mathbf{I}\mathbf{f} - \frac{2}{\lambda_{max}}\theta_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{f} - \theta_1\mathbf{I}\mathbf{f} \\ &= \theta_0\mathbf{I}\mathbf{f} - \frac{2}{\lambda_{max}}\theta_0\mathbf{I}\mathbf{f} + \frac{2}{\lambda_{max}}\theta_0\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{f} + \theta_0\mathbf{I}\mathbf{f} \\ &= \frac{2\lambda_{max} - 2}{\lambda_{max}}\theta_0\mathbf{I}\mathbf{f} + \frac{2}{\lambda_{max}}\theta_0\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{f}\end{aligned} \quad (3)$$

In Eq. 3, the first and second term could be regarded as the all-pass filter with weight $\frac{2\lambda_{max}-2}{\lambda_{max}}$ and low-pass filter with weight $\frac{2}{\lambda_{max}}$, respectively. With these settings, we have the following theorem with theoretical analysis later.

Theorem 1. *While conducting spectral modulation, we can control the balance between all-pass and low-pass filters by simply tuning* λ_{max}. *And* $\lim_{\lambda_{max}\to\infty}\mathbf{f}^{'} \approx \theta_0\mathbf{I}\mathbf{f}$, *which is an all-pass filter;* $\lim_{\lambda_{max}\to 1}\mathbf{f}^{'} \approx \theta_0\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{f}$, *which is a low-pass filter.*

We can find the optimal threshold between low and high frequency signals by tuning the weights of these two filters. With a high λ_{max}, the weight of all-pass filters is elevated and so are the high frequency signals. Whereas when λ_{max} is low, the weight of low-pass filters is high and so are the low frequency signals. However, it is nontrivial to manually decide which part is more significant than the other as situations are different across various datasets. A natural step forward would be finding the optimal eigenvalues in a data-driven fashion. Hence, in order to find the optimal threshold, we make λ_{max} at k^{th} layer a learnable parameter:

$$\lambda_{max}^{k} = 1 + \text{relu}(\phi_k), \quad (4)$$

where $\phi_k \in \mathbb{R}$ is a learnable scalar, and relu(.) refers to rectified linear unit function. ϕ_k is initialized as 1, since $\lambda_{max}^{k} = 2$ when $\phi_k = 1$. Under this setting, the initial balance between two filters is identical (i.e., $\frac{2\lambda_{max}-2}{\lambda_{max}} = \frac{2}{\lambda_{max}} = 1$), preventing the model from being stuck at local-minimum. We regularize ϕ_k by a relu function because relu has a codomain of $[0, \infty]$. This enables the propagation layer to achieve a all-pass filter when $\phi_k \to \infty$ or a low-pass filter when $\phi_k \to 0$. Utilizing a layer-wise matrix representation, we have node embedding $\mathbf{H}^{(k)}$ at k^{th} layer as:

$$\begin{aligned}\mathbf{H}^{(k)} &= (\frac{2\lambda_{max}^{k} - 2}{\lambda_{max}^{k}}\mathbf{I} + \frac{2}{\lambda_{max}^{k}}\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{H}^{(k-1)}\mathbf{W}_k \\ &= (\frac{2\text{relu}(\phi_k)}{1 + \text{relu}(\phi_k)}\mathbf{I} + \frac{2}{1 + \text{relu}(\phi_k)}\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{H}^{(k-1)}\mathbf{W}_k,\end{aligned} \quad (5)$$

where $\mathbf{H}^{(0)} = \mathbf{X}$, $\mathbf{W}_k \in \mathbb{R}^{d^{(k-1)}\times d^{(k)}}$ denotes the parameter matrix of filter at k^{th} layer and $d^{(k)}$ refers to the dimension of signals at k^{th} layer. The domain of eigenvalues of $(\frac{2\text{relu}(\phi_k)}{1+\text{relu}(\phi_k)}\mathbf{I} + \frac{2}{1+\text{relu}(\phi_k)}\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})$ is [0, 2], which can introduce numerical instabilities and unexpected gradient issues. So using the renormalization trick proposed in (Kipf and Welling 2016), we further normalize and

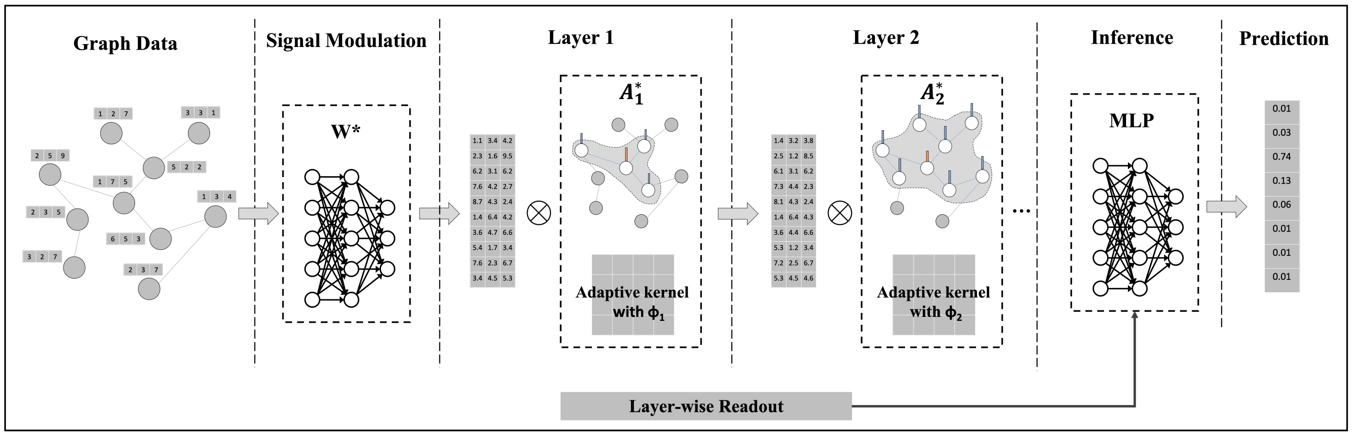

Figure 1: AKGNN for node classification. The parameters of filters at all layers are approximated by a single MLP. And at each propagation layer, AKGNN learns the optimal trade-off between all-pass and low-pass filters and constructs $\mathbf{A}_k^*$ to conduct graph convolution. The class label is inferred by summing node representations at all layers through a prediction MLP.

reformat $(\frac{2\text{relu}(\phi_k)}{1+\text{relu}(\phi_k)}\mathbf{I} + \frac{2}{1+\text{relu}(\phi_k)}\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})$ as $\mathbf{A}_k^* = \mathbf{D}_k^{-\frac{1}{2}}\mathbf{A}_k\mathbf{D}_k^{-\frac{1}{2}}$, where $\mathbf{A}_k = \frac{2\text{relu}(\phi_k)}{1+\text{relu}(\phi_k)}\mathbf{I} + \frac{2}{1+\text{relu}(\phi_k)}\mathbf{A}$, and $\mathbf{D}_k$ denotes the diagonal degree matrix of $\mathbf{A}_k$. Putting them together, the layer-wise propagation is summarized as:

$$\mathbf{H}^{(k)} = \mathbf{A}_k^*\mathbf{H}^{(k-1)}\mathbf{W}_k. \tag{6}$$

Many current GNNs (Klicpera, Bojchevski, and Günnemann 2019; Zhu and Goldberg 2009; Chen et al. 2020b) have adopted kernels where the balance between all-pass and low-pass filters are dedicatedly tailored. They utilize a pre-defined balancing variable to achieve so but finding the optimal threshold for a specific dataset is undeniably non-trivial as the search space is usually very large. Different from these current approaches, the adjacency matrix $\mathbf{A}_k^*$ we utilize at k^{th} layer is parameterized with a single scalar. This design enables our model to effectively learn the optimal balance between high and low frequency signals during the training phase and omit the cumbersome hyper-parameter tuning. However, it is difficult for the model to simultaneously learn both the filter $\mathbf{W}_k$ and the optimized graph Laplacian $\mathbf{A}_k^*$ since the filter operates on the current version of graph Laplacian and dynamically updating both might lead to a situation where the whole model will never converge. Moreover, as we stack numerous layers to capture the high-order information, $\mathbf{W}_k$ still introduces a number of parameters, which are very likely to introduce the over-fit issue. Hence we utilize a parameterization trick to alleviate the above issues.

Parameterization trick

The key motivation of all graph convolutions to stack multiple propagation layers is capturing high-order information that is beneficial to downstream tasks. As mentioned in the introduction, aiming to capture such information, under the designs of most GNNs, more parameters are also introduced (e.g., $\mathbf{W}_k$ in Eq. 6). This could bring the over-fitting problem when nodes are scarcely labeled and offset the intended benefits. (Wu et al. 2019) proposes to approximate parameters at all layers with a single matrix and meanwhile eliminate the non-linearity in between, which is proved to achieve similar results with fewer parameters. Nevertheless, by conducting such approximation, the complexity of the graph filter is also significantly decreased, making dynamically tuning both the filter and graph Laplacian feasible. Hence, we utilize a modified version of such parameterization trick to approximate parameters at all layers with a single matrix. Specifically, we can re-write Eq. 6 by expanding $\mathbf{H}^{(k-1)}$ as follows:

$$\begin{aligned}\mathbf{H}^{(K)} &= \mathbf{A}_k^*\mathbf{H}^{(K-1)}\mathbf{W}_K \\ &= \mathbf{A}_K^*\mathbf{A}_{K-1}^*\dots\mathbf{A}_1^*\mathbf{X}\mathbf{W}_1\dots\mathbf{W}_{K-1}\mathbf{W}_K,\end{aligned} \tag{7}$$

where K is the total number of propagation layers. We propose to use a single matrix $\mathbf{W}^* \in \mathbb{R}^{d\times d^{(K)}}$ to approximate the functionalities of all $\mathbf{W}_k$, such that:

$$\mathbf{H}^{(k)} = \mathbf{A}_k^*\mathbf{H}^{(k-1)} \quad \text{for } k \geq 1, \text{and } \mathbf{H}^{(0)} = \sigma(\mathbf{X}\mathbf{W}^*), \tag{8}$$

where $\sigma(.)$ denotes the ReLU non-linearity. From the perspective of spatial aggregation, intuitively, i^{th} row of $\mathbf{H}^{(k)}$ is simply a linear combination of the modulated graph signals of node v_i and those of its neighbors, whose distances to v_i are within k hops. Through this trick, each convolution layer has only one parameter ϕ_k, which also significantly alleviates the convergence issue.

Inference and Prediction

After performing signal modulation for K propagation layers, we generate K node representation matrices $\{\mathbf{H}^{(k)}|1 \leq k \leq K\}$. These matrices are combined through a readout function and then fed into a Multi-layer Perceptron (MLP) to predict the class labels $\mathbf{Y}^P \subseteq \{0,1\}^{N\times C}$, formulated as:

$$\mathbf{Y}^P = \text{softmax}(f_{MLP}(READOUT(\mathbf{H}^{(k)}))), \tag{9}$$

where $READOUT(.)$ denotes the layer-wise readout function. We choose to combine intermediate node representations through a readout function, instead of using $\mathbf{H}^{(k)}$ directly for the final prediction, because it is very possible that different nodes require distinct levels of information for node classification. And bringing high-order information for nodes whose labels could be inferred merely through local information might jeopardize the decision margin (Xu et al. 2018b; Liu, Gao, and Ji 2020). As for the selection of readout function, we explore element-wise summation $sum(.)$ instead of other element-wise operations (e.g., mean, or max) to maximize the express power (Xu et al. 2018a). Compared with other readout functions, the summation function is injective and with such function, we reduce the possibility of nodes, that share the same graph signal coefficients but are structurally different, being represented the same. Layer-wise concatenation is also able to achieve similar functionalities but it also introduces the number of parameters in $f_{MLP}(.)$.

Theoretical Analysis

In this section, we explain the mechanism of adaptive kernel learning with maximal eigenvalue from the perspective of spectral graph theory. Recall that graph Laplacian is formulated as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ or its normalized form $\mathbf{L} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{A})\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. The Laplacian matrix of either form is symmetric and semi-positive definite, which gives it an important property: it can be eigen-decomposed such one set of resulted eigenvalues are all greater than or equal to zero, formulated as $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$. We sort the eigenvalues and their corresponding eigenvectors such that $0 = \lambda_1 \leq \lambda_2 \cdots \leq \lambda_{N-1} \leq \lambda_N$. The key idea behind graph spectral filtering is modulating the graph signals' frequencies so that these beneficial to downstream tasks are magnified while others are diminished. This could be achieved by modifying the eigenvalues of Laplacian matrix with a filter function $f(.)$. (Defferrard, Bresson, and Vandergheynst 2016) proposes to utilize model $f(.)$ by Chebyshev polynomials $T_k(.)$. In the work of (Defferrard, Bresson, and Vandergheynst 2016), it is prerequisite that polynomials at different orders are orthogonal, because orthogonality guarantees that modifying filter at a specific order won't interfere with other orders. So it is necessary to normalize $\mathbf{\Lambda}$ as $\tilde{\mathbf{\Lambda}} = \frac{2\cdot\mathbf{\Lambda}}{\lambda_{max}} - \mathbf{I}$ such that its domain aligns with the domain [-1, 1] where the orthogonality of Chebyshev polynomials is defined. Under this setup, in order to modulate signals, at least $\mathcal{O}(d \times K)$ parameters are needed, where d, K stands for the dimension of input signals and the order of truncated polynomials, respectively. To reduce the complexity of this process, we propose to make λ_{max} as a learnable parameter. λ_{max} as a single parameter could effectively solve one major task of the graph filter: balancing the trade-off between high and low frequencies. When λ_{max} is large (e.g., $\lim_{\lambda_{max}\to\infty}$), all values on the diagonal of $\mathbf{\Lambda}$ becomes infinitely close to each other and hence every signal in the original graph Laplacian is retained, corresponding to the all-pass filter $\lim_{\lambda_{max}\to\infty}\mathbf{f}' \approx \theta_0\mathbf{I}\mathbf{f}$ in Theorem 1. Notice that the domain of normalized Laplacian matrix is upper-bounded by 2 and we allow the maximum eigenvalue to freely vary between $(1, \inf)$. In this case, our goal here is not to find the actual maximum eigenvalue; instead, we aim to utilize this normalization process such that the trade-off between all-pass and low-pass filters is optimized. If λ_{max} is upper-bounded by 2, in circumstances where high-frequency signals are significant for downstream tasks, the best scenario we can possibly approach is the equal weight on all-pass and low-pass filters (i.e., $\frac{2\lambda_{max}-2}{\lambda_{max}}\theta_0\mathbf{I}\mathbf{f} + \frac{2}{\lambda_{max}}\theta_0\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{f} = \mathbf{I}\mathbf{f} + \theta_0\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{f}$).

On the other hand, when λ_{max} is small (e.g., $\lim_{\lambda_{max}\to 1}$), we will have some high frequency signals whose eigenvalues are larger than λ_{max}. In this case, these signals will become unorthogonal to low frequency signals whose eigenvalues are less than or equal to λ_{max} and these high frequency signals can be generated by linear combinations of the low ones, which corresponds to low-pass filter $\lim_{\lambda_{max}\to 1}\mathbf{f}' \approx \theta_0\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{f}$. With the help of learnable λ_{max}, the complexity of graph signal filtering in AKGNN is significantly reduced. Because the scope of signal sets is diminished by learnable λ_{max} and filter only needs to focus on signals beneficial to the downstream task.

Complexity Analysis

The complexity of AKGNN can be decoupled into two parts. The first portion is graph signal filtering with $f_{MLP}(.)$, with complexity $\mathcal{O}(Nd \cdot d^{(K)})$, where N stands for the number of nodes, d refers to the dimension of the input signal, and $d^{(K)}$ is the dimension of the filtered signal. The second portion is graph convolution with the adaptive kernel, with complexity $\mathcal{O}(|E|d^{(K)})$ per each layer, where $|E|$ denotes the number of edges. Hence for a K-layer AKGNN, the total computational complexity is $\mathcal{O}(N \cdot d \cdot d^{(K)} + K \cdot |E| \cdot d^{(K)})$, which is linear with the number of nodes, edges, and layers.

Experiments and Analysis

We follow the experiment setup acknowledged as community conventions, the same as node classification tasks in (Kipf and Welling 2016; Veličković et al. 2017) (e.g., publicly fixed 20 nodes per class for training, 500 nodes for validation, and 1,000 nodes for testing). The three datasets we evaluate are Cora, Citeseer and Pubmed (Sen et al. 2008).

Baselines. We compare AKGNN with three GNN branches:

- ***Graph convolutions***: ChebyNet (Defferrard, Bresson, and Vandergheynst 2016), GCN (Kipf and Welling 2016), GAT (Veličković et al. 2017), JKNets (Xu et al. 2018b), APPNP (Klicpera, Bojchevski, and Günnemann 2019), SGC (Wu et al. 2019) and SSGC (Zhu and Koniusz 2021).
- ***Regularization based***: VBAT (Deng, Dong, and Zhu 2019), Dropedge (Rong et al. 2020), GAugO (Zhao et al. 2020), and PGSO (Dasoulas, Lutzeyer, and Vazirgiannis 2021). The backbone model used is GCN.
- ***Sampling based***: GraphSage (Hamilton, Ying, and Leskovec 2017), FastGCN (Chen, Ma, and Xiao 2018).

Method	Graph		
	Cora	Citeseer	Pubmed
ChebyNet	81.2	69.8	74.4
GCN	81.5	70.3	79.0
GAT	83.0±0.7	72.5±0.7	79.0±0.3
APPNP	83.8±0.3	71.6±0.5	79.7±0.3
SGC	81.0±0.0	71.9±0.1	78.9±0.0
SSGC	83.5±0.0	73.0±0.1	80.2±0.0
JKNets	83.3±0.5	72.6±0.3	79.2±0.3
PGSO	82.5±0.3	71.8±0.2	79.3±0.5
VBAT	83.6±0.5	73.1±0.6	79.9±0.4
GAugO	83.6±0.5	73.3±1.1	80.2±0.3
Dropedge	82.8	72.3	79.6
GraphSage	78.9±0.6	67.4±0.7	77.8±0.6
FastGCN	81.4±0.5	68.8±0.9	77.6±0.5
AKGNN (ours)	**84.4±0.3**	**73.5±0.2**	**80.4±0.3**
w/o λ learning	81.4±0.2	71.9±0.1	79.1±0.2
w/o PT	83.1±0.1	72.2±0.5	80.1±0.3
w/o readout	83.5±0.2	73.1±0.3	79.4±0.2

Table 1: Overall classification accuracy (%).

Implementation Detail We utilize PyTorch as our deep learning framework to implement AKGNN. The adaptive kernel learning mechanism is engineered in a sparse tensor fashion for compact memory consumption and fast back-propagation. The weights are initialized with Glorot normal initializer (Glorot and Bengio 2010). We explore Adam to optimize parameters of AKGNN with weight decay and use early stopping to control the training iterations based on validation loss. Besides, we also utilize dropout mechanism between all propagation layers. All the experiments in this work are implemented on a single NVIDIA GeForce RTX 2080 Ti with 11 GB memory size and we didn't encounter any memory bottleneck issue while running all experiments.

Hyperparameter Detail In AKGNN, related hyperparameters are the number of layers K, hidden size $d^{(K)}$, dropout rate, learning rate, weight decay rate, and patience for early stopping. We utilize identical hyperparameter settings over all three datasets as our model learns to adapt to different dataset. The number of layers K is 5, the hidden size $d^{(K)}$ is 64, the dropout rate between propagation layers is 0.6, the learning rate is 0.01, the weight decay rate is 5e-4, and patience for early stopping is 100 iterations.

Overall Results

From the upper portion of Tab. 1, we observe that AKGNN consistently outperforms all baselines by a noticeable margin over all datasets. Comparing AKGNN with the best-performing baseline of each dataset, we further conduct t-test and the improvement margin is statistically significant with the p-values less than 0.001. The improvement of AKGNN over GCN is 3.2%, 3.7% and 1.4% on Cora, Citeseer and Pubmed; whereas that over GAT is 1.4%, 1.3% and 1.4%. Compared with JKNet that utilizes a similar global readout function and parameterization trick, AKGNN has 1.1%, 0.9% and 1.2% improvements, demonstrating the efficacy of adaptive kernel learning mechanism. As the graph regularization-based model gets more attention, we also compare AKGNN with these recent works and the improvements are 0.8%, 0.2% and 0.2%.

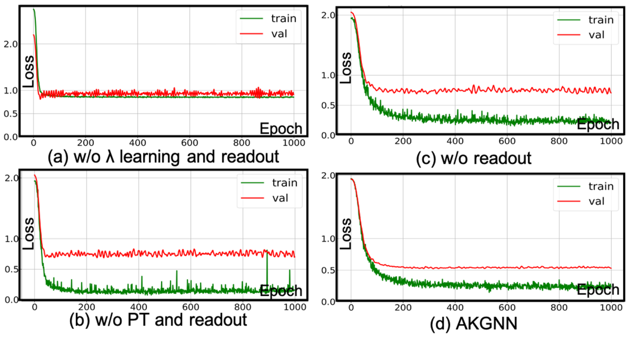

Figure 2: Generalization on Cora.

Ablation Studies

To analyze the contribution of different components in AKGNN, we conduct several sets of ablation studies. In order to examine the effectiveness of λ_{max} learning, we design the first variant as our model without adaptive kernel learning, denoted as 'w/o λ'. Another variant is our model without parameterization trick, denoted as 'w/o PT', aimed at validating its effectiveness in combating the over-fitting issue. The last variant is our model without readout function (i.e., $sum(.)$), in order to prove that nodes require different levels of information to achieve better performance. From the bottom portion of Tab. 1, we can first observe that all components contribute to the performance of AKGNN. The first variant without adaptive kernel learning and readout experiences a significant performance downgrade and is worse than the vanilla GCN model on some datasets, because we stack a lot of convolution layers and it encounters the over-smoothing. Comparing AKGNN without readout function with baselines, we observe similar performance.

In Fig. 2.a, both training and validation loss of this variant are highest and the differences between them are smallest across all variants, indicating that the over-smoothing issue has caused node representations to be indistinguishable. The second variant without parameterization trick has the lowest training loss, and also the biggest gap between training and validation loss, as shown in Fig. 2.b. This phenomenon represents the model suffers from the over-fitting problem due to the large number of parameters brought by numerous propagation layers. The third variant without readout function relatively performs better than the previous two, but still not as good as AKGNN, as shown in Fig. 2.c. This illustrates that the decision boundary is diminished as a result of bringing redundant high-order information for nodes that requires only low-frequency signals. Nevertheless, we further examine the generalization ability of our proposed method. As shown in Fig. 2.d, we can observe the lowest validation loss across all variants and meanwhile the differences between training and validation loss remain small, which demonstrates the generalization ability of AKGNN.

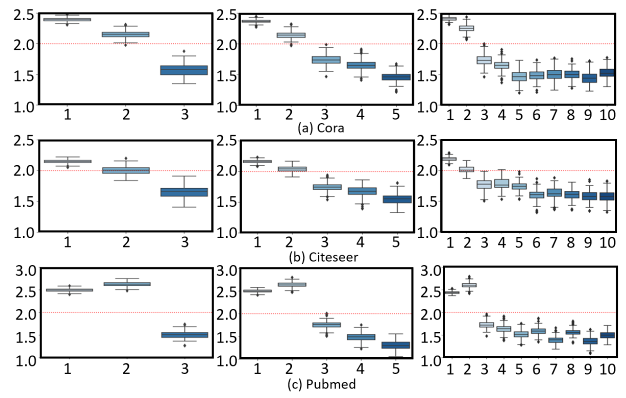

Figure 3: Maximal eigenvalues vs. number of layers (x-axis: layer k, y-axis: λ_{max}^k). Red dashed line indicates the equal weights of all-pass and low-pass filters. A higher λ_{max}^k indicates a higher weight of all-pass filter; whereas a low λ_{max}^k indicates a higher weight of low-pass filter.

Analysis of Adaptive Kernel Learning

The key motivation of learning the maximal eigenvalue is learning the optimal threshold between low and high frequency signals. In order to examine how it combats against the generality issue, we visualize the maximal eigenvalue at each layer, as shown in Fig. 3. We first analyze the dynamics of maximal eigenvalues λ_{max}^k within the same dataset. We can notice that the value of λ_{max}^k incrementally decreases as k progresses and reaches a plateau where λ_{max}^k doesn't change much after fifth layer across all datasets. We interpret this phenomenon as our model enforcing high-order layers to become meaningless. Because a low maximal eigenvalue at high-order layer would make node representations getting more indistinguishable. Moreover, λ_{max}^k at early layers doesn't deviate as K increases, demonstrating the strong contribution of local information and stability of AKGNN. Then we analyze the dynamics between these three datasets. We can notice that the λ_{max}^k of Pubmed has a higher mean than those of other two, showing that for node classification, high frequency signals benefits most for Pubmed dataset. Meanwhile, we can observe a significant λ_{max}^k drop at the second layer for Pubmed, indicating the redundancy of high-order information. This phenomenon also aligns with GNNs like GCN or GAT performing best with only two layers. Besides an intuitive explanation given above, we also theoretically explicate these phenomenon. Adapted Laplacian matrices across all layers share the same eigenvectors $\mathbf{U}$, because essentially our operation only modifies the diagonal eigenvalue matrix $\mathbf{\Lambda}$. Hence the commutative rule for matrix multiplication applies for all $\mathbf{A}_k^*$ as they are simultaneously diagonalizable and the order of $\mathbf{A}_k^*$ multiplication in Eq. 7 could be switch. In short, higher learned maximum eigenvalues should be observed if high-frequency signals dominate; whereas lower ones should be observed if low-frequency signals dominate. Across these three datasets, we can observe that AKGNN learns relatively high maximum eigenvalues compared with Cora and Pubmed, which aligns with the homophily property of there three datasets (i.e., Citeseer has the lowest homophily ratio.).

Analysis of Number of Propagation Layers

Beside adaptation of global readout function that has been utilized in (Liu, Gao, and Ji 2020; Xu et al. 2018b), our adaptive kernel learning mechanism also improves AKGNN's resilience to the over-reaching by enforcing high-order layers to become meaningless, as discussed in the previous sections. And the impact of the number of layers on performance is shown in Fig. 4. From this figure, we can notice that the accuracy on both testing and training reaches their highest around fifth layer and remains stable as the number of layers increases, demonstrating the AKGNN's strong resilience to the over-smoothing while the model is the over-reaching. We don't conduct experiments on AKGNN with more than 10 layers because 10-hop sub-graph of a target node covers almost all its possibly reachable nodes; the resilience to the over-fitting of AKGNN is only a byproduct of adaptive kernel learning and not the focus of this work.

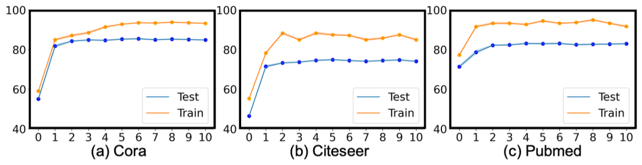

Figure 4: Impact of the number of layers on accuracy. (x-axis: K, y-axis: accuracy (%))

Conclusion

In this work, we study the problem of node representation learning on graphs and present Adaptive Kernel Graph Neural Network (AKGNN). In AKGNN, we propose adaptive kernel learning to find the optimal threshold between high and low frequency signals. Together with parameterization trick and global readout function, AKGNN is highly scalable, achieves competitive performance and retains so even with a number of convolution layers. Through experiments on three acknowledged benchmark datasets, AKGNN outperforms all baselines. Different from other graph convolution models whose guiding kernels are fixed and not ideal to all kinds of graphs, AKGNN learns to adapt to different graph Laplacians, which could shed light on a different path while researchers design new GNN models. We do not observe ethical concern or negative societal impact entailed by our method. However, care must be taken to ensure positive and societal consequences of machine learning. In the future, we aim to transfer the similar ideas of AKGNN to directed graphs or investigating the possibility of applying adaptive kernel learning to other kernels.

Acknowledgement

This work is partially supported by the NSF under grants IIS-2107172, IIS-2140785, CNS-1940859, CNS-1814825, IIS-2027127, IIS-2040144, IIS-1951504 and OAC-1940855, the NIJ 2018-75-CX-0032.

References

Ando, R. K.; and Zhang, T. 2007. Learning on graph with Laplacian regularization. *Advances in neural information processing systems*, 19: 25.

Chen, D.; Lin, Y.; Li, W.; Li, P.; Zhou, J.; and Sun, X. 2020a. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 3438–3445.

Chen, J.; Ma, T.; and Xiao, C. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*.

Chen, M.; Wei, Z.; Huang, Z.; Ding, B.; and Li, Y. 2020b. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, 1725–1735. PMLR.

Dasoulas, G.; Lutzeyer, J.; and Vazirgiannis, M. 2021. Learning Parametrised Graph Shift Operators. *arXiv preprint arXiv:2101.10050*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*.

Deng, Z.; Dong, Y.; and Zhu, J. 2019. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192*.

Dong, X.; Thanou, D.; Frossard, P.; and Vandergheynst, P. 2016. Learning Laplacian matrix in smooth graph signal representations. *IEEE Transactions on Signal Processing*, 64(23): 6160–6173.

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256. JMLR Workshop and Conference Proceedings.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. *arXiv preprint arXiv:1810.05997*.

Liu, M.; Gao, H.; and Ji, S. 2020. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 338–348.

Ma, Y.; Liu, X.; Zhao, T.; Liu, Y.; Tang, J.; and Shah, N. 2020. A unified view on graph neural networks as graph signal denoising. *arXiv preprint arXiv:2010.01777*.

Oono, K.; and Suzuki, T. 2019. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*.

Pang, J.; and Cheung, G. 2017. Graph Laplacian regularization for image denoising: Analysis in the continuous domain. *IEEE Transactions on Image Processing*, 26(4): 1770–1785.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. *arXiv preprint arXiv:1907.10903*.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine*, 29(3): 93–93.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, G.; Ying, R.; Huang, J.; and Leskovec, J. 2019. Improving graph attention networks with large margin-based constraints. *arXiv preprint arXiv:1910.11945*.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*, 6861–6871. PMLR.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018a. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018b. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, 5453–5462. PMLR.

Zhao, T.; Liu, Y.; Neves, L.; Woodford, O.; Jiang, M.; and Shah, N. 2020. Data Augmentation for Graph Neural Networks. *arXiv preprint arXiv:2006.06830*.

Zhu, H.; and Koniusz, P. 2021. Simple spectral graph convolution. In *International Conference on Learning Representations*.

Zhu, M.; Wang, X.; Shi, C.; Ji, H.; and Cui, P. 2021. Interpreting and Unifying Graph Neural Networks with An Optimization Framework. *arXiv preprint arXiv:2101.11859*.

Zhu, X.; and Goldberg, A. B. 2009. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1): 1–130.