How Powerful are Spectral Graph Neural Networks

Xiyuan Wang [1] **Muhan Zhang** [1 2]

Abstract

Spectral Graph Neural Network is a kind of Graph Neural Network (GNN) based on graph signal filters. Some models able to learn arbitrary spectral filters have emerged recently. However, few works analyze the expressive power of spectral GNNs. This paper studies spectral GNNs' expressive power theoretically. We first prove that even spectral GNNs without nonlinearity can produce arbitrary graph signals and give two conditions for reaching universality. They are: 1) no multiple eigenvalues of graph Laplacian, and 2) no missing frequency components in node features. We also establish a connection between the expressive power of spectral GNNs and Graph Isomorphism (GI) testing, the latter of which is often used to characterize spatial GNNs' expressive power. Moreover, we study the difference in empirical performance among different spectral GNNs with the same expressive power from an optimization perspective, and motivate the use of an orthogonal basis whose weight function corresponds to the graph signal density in the spectrum. Inspired by the analysis, we propose JacobiConv, which uses Jacobi basis due to its orthogonality and flexibility to adapt to a wide range of weight functions. JacobiConv deserts nonlinearity while outperforming all baselines on both synthetic and real-world datasets.

1. Introduction

Graph Neural Networks (GNNs) have achieved state-of-the-art performance on almost all tasks among various graph representation learning methods (Yao et al., 2019; Fout et al., 2017; Chen et al., 2018). Spectral GNNs are a kind of GNNs that design graph signal filters in the spectral domain. Though various models have emerged, spectral GNNs' expressive power is still under-researched. Moreover, these models differ mainly in the basis choices of the spectral filters; however, to our best knowledge, no study has systematically explained these differences and studied the advantages and disadvantages of different bases.

Existing spectral GNNs can be summarized into a general form: first transforming the spatial signal X through an MLP, then applying spectral filters parameterized by a polynomial of the normalized Laplacian $\hat{L}$, and finally applying another MLP to the filtered signal. By designing/learning the polynomial coefficients, spectral GNNs can simulate a wide range of filters (low-pass, band-pass, high-pass) in the spectral domain, enabling GNNs to work on not only homophilic but also heterophilic graphs (Chien et al., 2021).

However, a natural question is whether the MLPs or nonlinearity are useful at all, or are spectral filters enough? To study this problem, we remove nonlinearity from spectral GNNs and explore the expressive power of such linear spectral GNNs whose power relies only on the spectral filters. We prove that linear GNNs are *universal* under some mild conditions, i.e., they are powerful enough to produce arbitrary predictions without relying on MLPs. Our results show that nonlinearity is unnecessary for spectral GNNs to reach high expressiveness, which is also verified in our experiments. Moreover, we analyze spectral GNNs' universality conditions from a Graph Isomorphism (GI) testing perspective. The latter is often used to characterize spatial GNNs' expressive power (Xu et al., 2019). Our results, for the first time, build a bridge between the expressivity analyses of spectral GNNs and spatial GNNs.

Next, we notice that spectral GNNs with different polynomial bases of the spectral filters have the same expressive power but different empirical performance. To study this difference, we analyze the optimization of such models. By checking the Hessian matrix of linear spectral GNNs near the global minimum, we find that using an orthogonal basis with the density of graph signal as the weight function can maximize the convergence speed.

Inspired by these discussions, we propose a novel expressive spectral GNN, JacobiConv. JacobiConv deserts nonlinearity, approximates filter functions with Jacobi basis, and is flexible enough in weight function choices to adapt to a

[1] Institute for Artificial Intelligence, Peking University [2] Beijing Institute for General Artificial Intelligence. Correspondence to: Muhan Zhang <muhan@pku.edu.cn>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

wide range of graph signal densities. We also design a novel Polynomial Coefficient Decomposition (PCD) technique to improve the filter coefficient optimization. In numerical experiments, we first test the expressive power of JacobiConv to approximate filter functions on synthetic datasets. JacobiConv achieves the lowest loss on learning the filter functions compared to state-of-the-art spectral GNNs. We also show that JacobiConv outperforms all baselines on ten real-world datasets by up to 12%.

2. Preliminaries

For any matrix $M \in \mathbb{R}^{a\times b}$, M_i is the i^{th} row of M, $M_{:i}$ is the i^{th} column of M, $M_{\mathbb{AB}}$ is the submatrix of M corresponding to row index set $\mathbb{A}$ and column index set $\mathbb{B}$. Let δ_{ij} denote Kronecker delta: 1 if $i=j$, and 0 otherwise. We define *condition number* of a matrix M as $\kappa(M)=\frac{|\lambda_{\text{max}}|}{|\lambda_{\text{min}}|}$, where λ_{min}, λ_{max} are the minimum and maximum eigenvalues of M, respectively. If M is singular, $\kappa(M)=+\infty$.

Let $\mathcal{G}=(\mathbb{V},\mathbb{E},X)$ denote an undirected graph with a finite node set $\mathbb{V}=\{1,2,...,n\}$, an edge set $\mathbb{E}\subseteq\mathbb{V}\times\mathbb{V}$ and a node feature matrix $X\in\mathbb{R}^{n\times d}$, whose i^{th} row X_i is the feature vector of node i. $N(i)$ refers to the set of nodes adjacent to node i. Let A be the adjacency matrix of $\mathcal{G}$ and D be the diagonal matrix whose diagonal element D_{ii} is the degree of node i. The normalized adjacency matrix is $\hat{A}=D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. Let I denote the identity matrix. The *normalized Laplacian matrix* $\hat{L}=I-\hat{A}$. Let $\hat{L}=U\Lambda U^T$ denote the eigendecomposition of $\hat{L}$, where U is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues.

2.1. Graph Isomorphism

A *permutation* π is a bijective mapping from $\{1,2,...,n\}$ to $\{1,2,...,n\}$, where $n\in\mathbb{N}^+$. For node set $\mathbb{V}$, $\pi(\mathbb{V})=\{\pi(i)|i\in\mathbb{V}\}$. For node feature matrix X, $\pi(X)_{\pi(i)}=X_i$. For edge set $\mathbb{E}$, $\pi(\mathbb{E})=\{(\pi(i),\pi(j))|(i,j)\in\mathbb{E}\}$. An *automorphism* of a graph $\mathcal{G}=(\mathbb{V},\mathbb{E})$ is a permutation π such that $\pi(\mathbb{V})=\mathbb{V},\pi(\mathbb{E})=\mathbb{E}$. An automorphism of a graph with node features $\mathcal{G}=(\mathbb{V},\mathbb{E},X)$ is a permutation π such that $\pi(\mathbb{V})=\mathbb{V},\pi(\mathbb{E})=\mathbb{E},\pi(X)=X$. The *order* of an automorphism is $\min_k \pi^k=e, k=1,2,...$, where e is the identity mapping. Two nodes i,j are *isomorphic* if $\pi(i)=j$ under some automorphism π.

2.2. Graph Signal Filter and Spectral GNNs

The *graph Fourier transform* of a signal $X\in\mathbb{R}^{n\times d}$ is defined as $\tilde{X}=U^TX\in\mathbb{R}^{n\times d}$. The *inverse transform* is $X=U\tilde{X}$ (Shuman et al., 2013). The i^{th} column of U is a frequency component corresponding to the eigenvalue λ_i.

Let $\tilde{X}_\lambda=U_{:\lambda}^TX$, where $U_{:\lambda}$ is the eigenvector corresponding to λ, be the frequency component of X at λ frequency. If $\tilde{X}_\lambda\neq\vec{0}$, we say X contains the λ frequency component. Otherwise, the λ frequency component is missing from X.

We can use $g:[0,2]\to\mathbb{R}$ to filter each frequency component by multiplying $g(\lambda)$. Applying a spectral filter g on signal X is defined as follows

$$Ug(\Lambda)U^TX, \tag{1}$$

where $g(\Lambda)$ applies g element-wisely to the diagonal entries of Λ. To parameterize the filter, g is often set to be a polynomial of degree K

$$g(\lambda):=\sum_{k=0}^{K}\alpha_k\lambda^k. \tag{2}$$

Then, the filtering process can be expressed by

$$Ug(\Lambda)U^TX=\sum_{k=0}^{K}\alpha_kU\Lambda^kU^TX=\sum_{k=0}^{K}\alpha_k\hat{L}^kX. \tag{3}$$

By defining $g(\hat{L})=\sum_{k=0}^{K}\alpha_k\hat{L}^k$, we can rewrite the filtering process as follows:

$$Ug(\Lambda)U^TX=g(\hat{L})X. \tag{4}$$

We show the forms of some popular spectral GNNs in Appendix A. In general, existing spectral-based GNNs can be unified into the following form:

$$Z=\phi(g(\hat{L})\varphi(X)), \tag{5}$$

where Z is the prediction, ϕ, φ are functions like multi-layer perceptrons (MLPs) and g is a polynomial. If a spectral-based GNN can express any polynomial filter function g, we call it *Polynomial-Filter-Most-Expressive* (PFME) GNNs. We also define *Filter-Most-Expressive* (FME) GNNs as the GNNs able to express arbitrary real-valued filter functions.

This study mainly focuses on the case when ϕ and φ are linear functions, so we define *linear GNN*.

Definition 2.1. A **linear GNN** can be formulated as $Z=g(\hat{L})XW$, where $Z\in\mathbb{R}^{n\times d'}$ is the prediction matrix, g is a learnable real-valued polynomial, and $W\in\mathbb{R}^{d\times d'}$ is a learnable matrix.

Linear GNN keeps the spectral filter form of spectral GNNs despite its simplicity. And the expressive power of linear GNNs is a **lower bound** for that of general spectral GNNs in Equation (5).

Proposition 2.2. *Linear GNN is PFME. If ϕ and φ can express all linear functions, spectral GNNs can differentiate any pair of nodes which linear GNNs can differentiate.*

We assume all our models work on a fixed graph with fixed node features and only perform node property prediction tasks. Suppose there is an arbitrary real-valued filter function to approximate. Though PFME GNNs can only express polynomial filter functions, as the eigenvalue λ is a discrete variable in a fixed graph, an interpolation polynomial always exists for the arbitrary filter and can produce the same output (Burden & Faires, 2005). Therefore, PFME GNNs are FME in our problem setting. In the following analysis, we always assume the linear GNNs have a high-enough degree K in their polynomial filters. Linear GNNs with a limited degree K are discussed in Appendix C.

3. Related Work

3.1. Spectral GNNs

Spectral GNNs are GNNs based on spectral graph filters (Wu et al., 2021). He et al. (2021) categorize spectral GNNs by the filter operation adopted. One class is spectral GNNs with fixed filters: APPNP (Klicpera et al., 2019a) utilizes Personalized PageRank (PPR) (Page et al., 1999) to build filter functions. GNN-LF/HF (Zhu et al., 2021) designs filter weights from the perspective of graph optimization functions. The other class is spectral GNNs with learnable filters: ChebyNet (Defferrard et al., 2016) approximates the filters with Chebyshev polynomials. GPRGNN (Chien et al., 2021) learns a polynomial filter by directly performing gradient descent on the polynomial coefficients. ARMA (Bianchi et al., 2021) uses rational filters. BernNet (He et al., 2021) expresses the filtering operation with Bernstein polynomials. Though ARMA (Bianchi et al., 2021) and GNN-LF/HF (Zhu et al., 2021) use rational functions, they approximate the rational functions with polynomials. Therefore, all these methods use some form of polynomial filter despite the different bases they use. GPRGNN is one of the most expressive models. It can express all polynomial filters. So does ChebyNet, as the Chebyshev polynomials also form a complete set of bases in the polynomial space. They are both PFME. BernNet is less expressive as it forces the coefficients of the Bernstein polynomial bases to be positive and can only express positive filter functions. However, such constraints are introduced for regularization, so we ignore them when analyzing the expressive power. The filter forms of these models are summarized in Table 5.

3.2. Removing Nonlinearity from GNNs

Various GNNs removing nonlinearity have been proposed. Wu et al. (2019) precompute $\hat{A}^k X$ and perform logistic regression on the preprocessed features. Some works leverage personalized PageRank (PPR) (Page et al., 1999) and random walk on the graph. Klicpera et al. (2019b) use generalized graph diffusion, like the heat kernel and PPR, to reconstruct the graph. APPNP (Klicpera et al., 2019a) replaces normalized adjacency matrix with approximate PPR matrix to capture multi-hop neighborhood information. Some models with more complex acceleration techniques for computing PPR are introduced, like GBP (Chen et al., 2020a). Existing linear models are mainly motivated by improving the scalability and have restricted filters. In contrast, we analyze the expressive power and optimization property of linear GNNs with arbitrary polynomial filters.

3.3. Expressive Power of GNNs

The Weisfeiler-Lehman (WL) test of graph isomorphism (Weisfeiler & Leman, 1968) is a series of algorithms that can distinguish almost all non-isomorphic graphs. Its 1-dimensional form (1-WL) iteratively aggregates neighborhood labels and maps the aggregated labels into a new label for each node, which is similar to GNNs based on neighborhood node feature aggregation. The node labels assigned by 1-WL test can also be used to check if two nodes are isomorphic (Babai & Kucera, 1979; Srinivasan & Ribeiro, 2020), as two isomorphic nodes always have the same label while two non-isomorphic ones mostly have different labels. Xu et al. (2019) show that the 1-WL test bounds the expressive power of GNNs to distinguish non-isomorphic graphs. Since then, various works have attempted to analyze GNNs with the WL test and graph isomorphism testing (Zhang et al., 2018; Morris et al., 2019; Maron et al., 2019a; Chen et al., 2019; Li et al., 2020; Sato et al., 2019; 2021; Zhang et al., 2021; Zhang & Li, 2021). Other than the WL test and graph isomorphism, some works measure the expressive power in different ways, such as expressing universal invariant functions (Maron et al., 2019b; Keriven & Peyré, 2019), counting substructures (Chen et al., 2020b), simulating Turing Machine (Loukas, 2020), computing graph properties (Garg et al., 2020), and differentiating rooted graphs (Chen et al., 2021). Balcilar et al. (2021) analyze the expressive power from a spectral perspective, but the discussion is constrained to concluding former models to spectral forms. Our study provides conditions for spectral GNNs to approximate any functions and discuss the relation between these conditions and graph isomorphism.

4. Expressive Power of Linear GNNs

In this section, we prove that linear GNNs are universal under three conditions and discuss these conditions to characterize how powerful spectral GNNs can reach. All proofs are in the appendix.

There are two components in a linear GNN $Z = g(\hat{L})XW$:

Linear Transformation W. Since $XW = U(\tilde{X}W)$, a linear transformation in the spatial domain is also a linear transformation in the frequency domain, which produces a linear combination of signals in different channels.

Filter $g(\hat{L})$. Since $g(\hat{L})X = U(g(\Lambda)\tilde{X})$, the filtering operation scales the frequency component corresponding to λ of

$\tilde{X}$ by $g(\lambda)$ fold in the frequency domain.

Now we give the **universality theorem** of linear GNNs.

Theorem 4.1. *Linear GNNs can produce any one-dimensional prediction if $\hat{L}$ has no multiple eigenvalues and the node features X contain all frequency components.*

There are three conditions for linear GNNs to be universal: 1) one-dimensional prediction, 2) no multiple eigenvalues, and 3) no missing frequency components. These are thus three bottlenecks for linear GNNs' expressive power. In the following, we discuss each of the three conditions in detail.

4.1. Multidimensional Prediction

Though linear GNNs are powerful when the output has only one dimension, each dimension may need a different polynomial filter when the prediction has multiple channels. Take the toy graph in Figure 1 as an example. One output dimension filters out high frequency signal and maintains low frequency signal, while the other one does the opposite. Therefore, a low-pass filter is needed for the first output dimension while a high-pass filter is needed for the second dimension. Using the same filter for all output dimensions cannot achieve this purpose.

We formally describe this property in Proposition 4.2.

Proposition 4.2. *If the node feature matrix X is not a full-row-rank matrix, for all $k > 1$ and all graphs, there exists a k-dimensional prediction linear GNNs cannot produce.*

We can use individual polynomial coefficients to compose a different filter for each output channel to solve this problem.

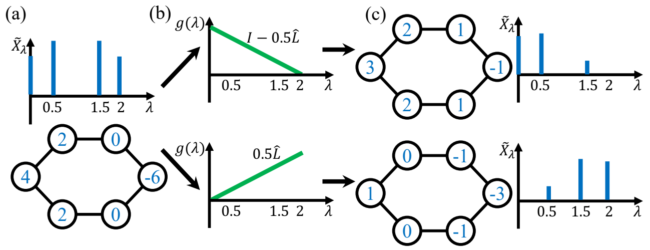

Figure 1. Individual filter function is needed for each prediction dimension. We illustrate each graph with both its spatial representation (where numbers on nodes represent one-dimensional node features) and its spectrum. (a) A graph with its node features. (b) Two different filters for two output dimensions. (c) Two output dimensions.

4.2. Multiple Eigenvalue

If two frequency components have the same eigenvalue λ, they will be scaled by the same number $g(\lambda)$. Therefore, the coefficients of these frequency components in prediction will keep the same ratio as in input XW. This issue is related to graph topology. More discussion is in Theorem 4.5.

4.3. Missing Frequency Components

The filter operation can only scale a frequency component. If this frequency component is missing from the node feature, the prediction cannot contain it either. Take the toy graph in Figure 2 as an example. The node features only contain component corresponding to frequency $\lambda = 0$, so a linear GNN cannot produce output with frequency $\lambda = 2$ component. This problem is rooted in both the topology of graph $\mathcal{G}$ and node features X and is difficult to solve.

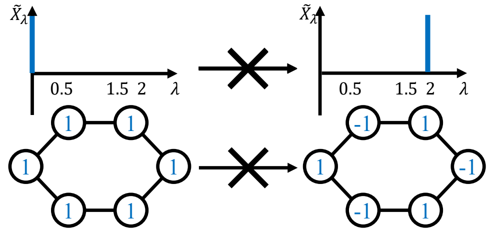

Figure 2. Node features with missing frequency components cannot produce some outputs.

Nevertheless, multiple eigenvalues and missing frequency components are both **rare in real-world graphs** with node features. See Appendix G for the ratio of multiple eigenvalues and number of missing frequency components in each of the 10 real-world benchmark datasets. In all the datasets, no frequency component is missing, and on average less than 1% of eigenvalues are multiple. Therefore, the universality conditions can be largely satisfied in practice.

4.4. Connection to Graph Isomorphism

Traditional expressivity analyses for spatial GNNs often leverage Graph Isomorphism testing. In this section, we explore the connections between our universality conditions and GI. We first build a connection between the expressive power of linear GNNs using a K-degree polynomial filter function and that of $(K+1)$-iteration WL test.

Proposition 4.3. *Given a linear GNN whose filter function is a K-degree polynomial, define the function $LG_K(i)$ as the prediction of node i produced by the linear GNN. Let $WL_k(i)$ denote the label of node i produced by k-iteration WL test whose initial label of node i is the node feature vector X_i. Then $\forall i, j \in \mathbb{V}$, $LG_K(i) = LG_K(j)$ if $WL_{K+1}(i) = WL_{K+1}(j)$.*

Proposition 4.3 means that linear GNNs' expressive power is also bounded by the 1-WL test: if 1-WL cannot differentiate two nodes, linear GNNs will also fail. However, this result seems to contradict with the universal approximation property of linear GNNs. We know that: 1) 1-WL provably cannot discriminate some non-isomorphic nodes (such as nodes in a non-attributed regular graph), and 2) 1-WL always gives isomorphic nodes the same label. However, a universal linear GNN should be able to give any two nodes different predictions, no matter whether they are isomorphic or not. To close this gap, we study the connections between the universality conditions of linear GNNs and the GI problem. Our results show that the no-multiple-eigenvalue and no-missing-frequency conditions enable 1-WL to discriminate all non-isomorphic nodes, and also constrain the graph to contain no isomorphic nodes, therefore closing the gap.

We first show that 1-WL can discriminate all non-isomorphic nodes under the two conditions.

Corollary 4.4. *If a graph has no missing frequency component and its normalized Laplacian has no multiple eigenvalues, then* 1*-WL can differentiate all non-isomorphic nodes.*

The other part of the gap is that 1-WL cannot produce different labels for isomorphic nodes, while linear GNNs with universal approximation property can. Therefore, we analyze how our no-multiple-eigenvalue and no-missing-frequency conditions constrain the graph in Theorems 4.5 and 4.6.

Theorem 4.5. *For a graph whose normalized Laplacian has no multiple eigenvalues, the order of its automorphism is less than three.*

The above theorem relates multiple eigenvalue to the degree of symmetry of the graph. A highly symmetric graph with three- or higher-order automorphism always has multiple eigenvalues. To intuitively understand this, we show two toy graphs in Figure 3. The triangle (b) has a three-order automorphism and an eigenvector can be permuted to produce another linearly independent eigenvector, while for the two-node graph (a) with only two-order automorphism, permuting its eigenvector results in the same eigenvector. Since real-world graphs are often highly irregular, Theorem 4.5 partly explains why multiple eigenvalues are rare in practice.

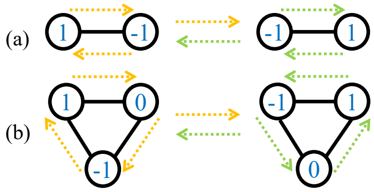

Figure 3. Graphs with high-order automorphisms (symmetries) have multiple Laplacian eigenvalues.

When considering node features containing all frequency components, all pairs of nodes are non-isomorphic, thus closing the gap between 1-WL and linear GNN.

Theorem 4.6. *Suppose a graph with node features does not have multiple eigenvalues in its normalized Laplacian, and no frequency component is missing from the node features. There will be no automorphism for the graph other than the identical mapping.*

Therefore, the conditions of Theorem 4.1 constrain the graph topology and node features so that 1-WL still bounds the expressive power of linear GNNs. On the other hand, our results indicate that 1-WL can be quite powerful given **expressive node features and irregular graph structures**. Our results build a bridge between the expressive power of spectral GNNs (in terms of universality under some conditions) and spatial GNNs (in terms of 1-WL test). As an analysis example, we also discuss how random features can boost the expressive power and why models with random features have poor empirical performance in Appendix D. Wang et al. (2022) also relates multiple eigenvalues to stability of positional encoding.

4.5. Role of Nonlinearity

Though linear GNNs have strong theoretical expressive power and remarkable empirical performance, various existing state-of-the-art GNNs utilize nonlinear activation functions. In this section, we analyze the role of nonlinearity.

In linear GNNs, the λ frequency component of the prediction $\tilde{Z}_\lambda$ is a function of only $g(\lambda)$, $\tilde{X}_\lambda$ and W. However, for nonlinear GNNs, different frequency components can transformed to each other. Figure 4 is an example, where new frequency components emerge after ReLU activation. Consider an element-wise activation function σ over the spatial signal X. We investigate its equivalent effect σ' over the spectral signal $\tilde{X}$. Its function on a spectral signal is $\sigma'(\tilde{X}) = U^T\sigma(U\tilde{X})$, meaning that different frequency components are first mixed via U, then nonlinearly transformed via σ element-wisely, and finally distributed back to each frequency via U^T. Thus, σ' is a column-wise nonlinear function over all frequency components. Mixing different frequency components may alleviate the issues from multiple eigenvalues and missing frequency components. However, such a mix is not expressive enough to solve all the problems, as 1-WL still bounds the expressive power of GNNs. Furthermore, since the universality conditions are easily satisfied by real-world graphs to a large degree (Appendix G), we desert nonlinearity in our experiments.

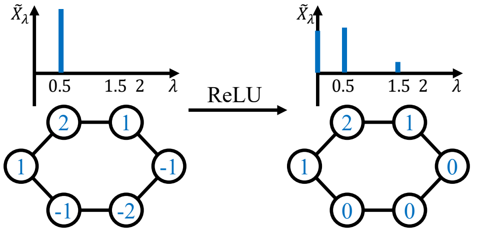

Figure 4. Nonlinear functions can mix different frequency components.

4.6. Role of Bias

Bias is usually used together with a linear transformation. However, there is no need to discuss bias separately because,

$$XW + b = \begin{bmatrix} X & 1_n \end{bmatrix} \begin{bmatrix} W \\ b \end{bmatrix}, \tag{6}$$

where row vector b is the learnable bias, 1_n is a column vector whose all elements are 1. Using bias is equivalent to adding 1 to the features of each node and thus still keeps the linear GNN form. Therefore, we ignore bias in theoretical analysis, but by default turn on bias in experiments for better performance. As bias introduces extra graph signals, a natural question is whether bias can complete the frequency components missing from original node features. The answer is no. Please see Appendix E for more details.

5. Choice of Basis for Polynomial Filters

Assume the polynomial bases are $g_k(\lambda), k = 0, 1, 2, ...$ In this section, we discuss linear GNNs with individual filter parameters for each output dimension, which is formulated as

$$Z_{:l} = \sum_{k=0}^{K} \alpha_{kl} g_k(\hat{L}) X W_{:l} \tag{7}$$

where α_{kl} is the coefficient of polynomial filter basis $g_k(\hat{L})$ and $(XW)_{:l}$ is the transformed node features for the l^{th} output dimension $Z_{:l}$.

All complete polynomial bases can build PFME models. However, models with different bases show different empirical performance. This section analyzes the effect of polynomial basis from an optimization perspective, which motivates the use of Jacobi Polynomials in our model.

5.1. Hessian Matrix and Polynomial Basis

Following the setting in (Xu et al., 2021), we study linear GNNs trained with the squared loss $R = \frac{1}{2}||Z - Y||_F^2$, where Y is the target. Assuming that linear GNNs can converge to the global minimum, we study the convergence speed near the global minimum. The rationality of this assumption is discussed in Appendix J.

When considering the optimization of a linear GNN, both α and W are learnable parameters. However, the gradient of loss over W is a function of the learnable filter function $g_{:l}(\hat{L}) := \sum_k \alpha_{kl} g_k(\hat{L})$ as a whole.

$$\frac{\partial R}{\partial W_{jl}} = \left[g_{:l}(\hat{L})(XW)_{:l} - Y_{:l}\right]^T \left[g_{:l}(\hat{L}) X_{:j}\right], \tag{8}$$

According to our assumption, the learned filter function is approximately the same for different bases as they have the same expressive power and can all converge to the global minimum. So the optimization of W is irrelevant to the choice of basis near the global minimum. However, the optimization of α heavily depends on the basis choice. To focus on the effect of basis choice, we only analyze the optimization of α by merging W into X.

Consider the optimization w.r.t. α. The loss is a convex function, and the gradient descent's convergence rate depends on the Hessian matrix's condition number (Boyd & Vandenberghe, 2009). Therefore, we analyze the Hessian matrix of linear GNNs near the global minimum.

Since the total loss is summed over different output dimensions, and each output dimension adopts a different set of polynomial coefficients α_{kl}, we can analyze the Hessian w.r.t. each dimension independently. Ignoring l, the (k_1, k_2) element of the Hessian matrix H can be written as

$$\begin{aligned}\frac{\partial R}{\partial \alpha_{k_1} \partial \alpha_{k_2}} &= X^T g_{k_2}(\hat{L}) g_{k_1}(\hat{L}) X \\ &= \sum_{i=1}^{n} g_{k_2}(\lambda_i) g_{k_1}(\lambda_i) \tilde{X}_{\lambda_i}^2.\end{aligned} \tag{9}$$

It can be equivalently expressed as a Riemann sum:

$$\sum_{i=1}^{n} g_{k_2}(\lambda_i) g_{k_1}(\lambda_i) \frac{F(\lambda_i) - F(\lambda_{i-1})}{\lambda_i - \lambda_{i-1}} (\lambda_i - \lambda_{i-1}), \tag{10}$$

where $F(\lambda) := \sum_{\lambda_i \le \lambda} \tilde{X}_{\lambda_i}^2$ is the accumulated amplitude of signal with frequency lower than λ. Define $f(\lambda) = \frac{\Delta F(\lambda)}{\Delta \lambda}$, which is the density of signal at frequency λ. In the limit when $n \to \infty$, we have:

$$H_{k_1 k_2} = \int_{\lambda=0}^{2} g_{k_1}(\lambda) g_{k_2}(\lambda) f(\lambda) \mathrm{d}\lambda. \tag{11}$$

The condition number $\kappa(H)$ reaches minimum if H is an identity matrix, which is equivalent to that g_k's form an orthonormal basis in the polynomial space whose inner product is defined by $\langle h, g \rangle = \int_0^2 h(\lambda) g(\lambda) f(\lambda) \mathrm{d}\lambda$ with $f(\lambda)$ being the weight function.

Our results show that although all complete polynomial bases have the same expressive power, using a set of orthonormal bases g_k whose weight function corresponds to the graph signal density can enable linear GNNs to achieve the highest convergence rate. As the normalization of bases is straightforward, we only consider orthogonality in the analysis.

Given the weight function $f(\lambda)$, we can construct an orthonormal basis using the Gram-Schmidt process. However, the exact form of the weight function f depends on the eigendecomposition of $\hat{L}$ and cannot be calculated efficiently and accurately for large graphs. Therefore, we choose a **general form** of orthogonal polynomials with **flexible enough** weight functions to adapt to different graph signal density functions $f(\lambda)$.

5.2. Jacobi Polynomial Bases

Among orthogonal polynomials, the Jacobi basis has a very general form, whereas the Chebyshev basis is a special case.

The Jacobi basis $P_k^{a,b}$ has the following form.

$$\begin{aligned} P_0^{a,b}(z) &= 1, \\ P_1^{a,b}(z) &= \frac{a-b}{2} + \frac{a+b+2}{2}z. \end{aligned} \tag{12}$$

For $k \geq 2$.

$$P_k^{a,b}(z) = (\theta_k z + \theta'_k)P_{k-1}^{a,b}(z) - \theta''_k P_{k-2}^{a,b}(z), \tag{13}$$

where

$$\begin{aligned} \theta_k &= \frac{(2k+a+b)(2k+a+b-1)}{2k(k+a+b)}, \\ \theta'_k &= \frac{(2k+a+b-1)(a^2-b^2)}{2k(k+a+b)(2k+a+b-2)}, \\ \theta''_k &= \frac{(k+a-1)(k+b-1)(2k+a+b)}{k(k+a+b)(2k+a+b-2)}. \end{aligned} \tag{14}$$

$P_k^{a,b}, k = 0, 1, 2, ...$ are orthogonal w.r.t. the weight function $(1-\lambda)^a(1+\lambda)^b$ on $[-1, 1]$. We can define the Jacobi basis for graphs as $g_k(\hat{L}) = P_k^{a,b}(I - \hat{L}) = P_k^{a,b}(\hat{A})$.

5.3. A Discussion on Popular Filter Bases

In this section, we compare three popular polynomial bases with the Jacobi Polynomial: Monomial $(1-\lambda)^k$, Chebyshev $P_k^{-1/2,-1/2}(1-\lambda)$, and Bernstein $\binom{K}{k}(1-\frac{\lambda}{2})^{K-k}(\frac{\lambda}{2})^k$. These bases are visualized in Appendix K.

For the Monomial basis, we can prove that it cannot be orthogonal on any weight function.

Proposition 5.1. *On any weight function $f(\lambda)$ which fulfils the requirements of the inner product, the Monomial basis is not orthogonal.*

Chebyshev basis is a particular case of Jacobi basis and is only orthogonal w.r.t. a specific weight function. In contrast, the Jacobi basis can adapt to a wide range of weight functions.

For non-orthogonal bases such as Bernstein, the Hessian matrix might not be diagonal, but a small condition number may still be achieved. In Appendix F, we build a connection between the condition number of polynomial regression's Gram matrix using basis $g_k, k = 0, 1, 2, .., K$ and that of linear GNNs' Hessian matrix. Therefore, some existing conclusions from polynomial regression basis choice can still be used. For example, existing studies show that the Bernstein basis can also achieve a lower condition number than the Monomial basis (Marco & Martinez, 2010). Though both Bernstein and Jacobi basis can outperform Monomial, Jacobi basis can perform better if the weight function of Jacobi basis well approximates the data distribution. Our experiments find that the Jacobi basis outperforms the Bernstein basis on both synthetic and real-world datasets.

6. JacobiConv Architecture

In this section, we describe our JacobiConv architecture. As the dimension of node features X is often much larger than that of the transformed features $\hat{X}$, we first feed X into a linear layer, $\hat{X} = XW + b$, with bias (see Section 4.6), and then filter $\hat{X}$. There are three techniques used in the filter: multiple filter functions, Jacobi basis, and a novel polynomial coefficient decomposition (PCD) technique.

6.1. Multiple Filters

Motivated by our analysis in Section 4.1, we adopt an individual filter function for each output dimension. The JacobiConv can be formulated as

$$Z_{:l} = \sum_{k=0}^{K} \alpha_{kl} P_k^{a,b}(\hat{A})\hat{X}_{:l}. \tag{15}$$

6.2. Computation of Jacobi Basis

With the recursion formula of Jacobi basis, we can compute all bases in $O(K)$ time and do K message passing operations.

$$\begin{aligned} P_0^{a,b}(\hat{A})\hat{X} &= \hat{X}, \\ P_1^{a,b}(\hat{A})\hat{X} &= \frac{a-b}{2}\hat{X} + \frac{a+b+2}{2}\hat{A}\hat{X}. \end{aligned} \tag{16}$$

For $k \geq 2$,

$$\begin{aligned} P_k^{a,b}(\hat{A})\hat{X} = \theta_k \hat{A} P_{k-1}^{a,b}(\hat{A})\hat{X} + \theta'_k P_{k-1}^{a,b}(\hat{A})\hat{X} \\ - \theta''_k P_{k-2}^{a,b}(\hat{A})\hat{X}. \end{aligned} \tag{17}$$

6.3. Polynomial Coefficient Decomposition

The filter function we construct can be formulated as $\sum_{k=0}^{K} \alpha_{kl} P_k^{a,b}$. We find that in real-world datasets α_{kl} gets smaller as k gets higher. As α_{kl}'s have different magnitudes, the optimization can be hard. So we decompose α_{kl} to $\beta_{kl}\prod_{i=1}^{k}\gamma_i$, where γ_i's are shared among different output channels. And we set $\gamma_i = \gamma' \tanh \eta_i$, which enforces $\gamma_i \in [-\gamma', \gamma']$. We call this technique Polynomial Coefficient Decomposition (PCD). We can modify the recursion formula to implement PCD.

$$\begin{aligned} P_k^{a,b}(\hat{A})\hat{X} = \gamma_k\theta_k \hat{A} P_{k-1}^{a,b}(\hat{A})\hat{X} + \gamma_k\theta'_k P_{k-1}^{a,b}(\hat{A})\hat{X} \\ - \gamma_k\gamma_{k-1}\theta''_k P_{k-2}^{a,b}(\hat{A})\hat{X}. \end{aligned} \tag{18}$$

7. Experiment

In this section, we first conduct experiments on synthetic datasets to examine JacobiConv's ability to express filter functions, and then test JacobiConv on real-world datasets. Our code is available at https://github.com/GraphPKU/JacobiConv.

Table 1. Average of sum of squared loss over 50 images.

	LOW	HIGH	BAND	REJECT	COMB
GPRGNN	0.4169	0.0943	3.5121	3.7917	4.6549
ARMA	1.8478	1.8632	7.6922	8.2732	15.1214
CHEBYNET	0.8220	0.7867	2.2722	2.5296	4.0735
BERNNET	0.0314	0.0113	0.0411	0.9313	0.9982
JACOBICONV	**0.0003**	**0.0011**	**0.0213**	**0.0156**	**0.2933**
MONOMIAL	2.4076	4.2411	10.8856	8.7031	10.5596
CHEBYSHEV	0.9227	2.3198	7.7751	6.0065	9.1191
BERNSTEIN	0.0110	0.0058	0.1517	0.1607	0.5705

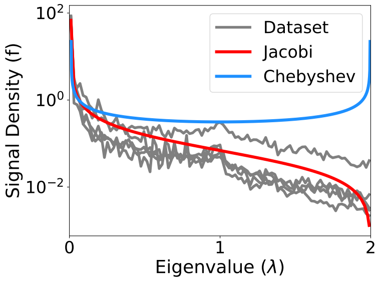

Figure 5. Signal density functions of some graphs in the image dataset and the weight functions of some bases.

7.1. Evaluating Models on Learning Filters

Following He et al. (2021), we transform real images to 2D regular 4-neighbor grid graphs, whose nodes are pixels. We apply 5 spectral filters (low $e^{-10\lambda^2}$, high $1 - e^{-10\lambda^2}$, band $e^{-10(\lambda-1)^2}$, reject $1 - e^{-10(\lambda-1)^2}$, and comb $|\sin \pi\lambda|$) to the signal in each image. All models use original graph signal as node features to fit the filtered signal.

We report the average squared error (lower the better) over the 50 pictures. Results are shown in Table 1. For a fair comparison, we remove PCD from linear GNNs.

We compare JacobiConv with popular PFME GNNs: GPRGNN (Chien et al., 2021), ARMA (Bianchi et al., 2021), BernNet (He et al., 2021), and ChebyNet (Defferrard et al., 2016). Settings of these models are detailed in Appendix H. JacobiConv outperforms other models on all datasets and even achieves up to 50 times lower loss on two datasets: Low and Reject. Though all these models can learn arbitrary polynomial filters, JacobiConv has better optimization properties as it uses orthogonal filter bases that can adapt to a wide range of signal distributions.

We also compare linear GNNs with different bases. The results are shown in the lower part of Table 1. Jacobi basis still outperforms other bases on all datasets and achieves 10 times lower loss than any other basis. Bernstein basis also achieves lower loss than Monomial on all datasets, which verifies our analysis in Section 5.3.

To verify that Jacobi basis adapts to the dataset, we plot the signal distributions of some randomly selected graphs in our image dataset and the weight functions of Jacobi and Chebyshev bases in Figure 5. We can see that only Jacobi basis (with hyperparameters selected for minimizing loss) can capture the main shape of the signal distribution, compared to Chebyshev basis.

Experimental results of models with PCD on synthetic datasets are shown in Appendix I. JacobiConv still outperforms any other model on all datasets. Jacobi basis also achieve a higher convergence rate than other bases for linear GNN. See Appendix L for the convergence rate.

7.2. Evaluation on Real-World Datasets

For homogeneous graphs, we include three citation graph datasets, Cora, CiteSeer and PubMed (Yang et al., 2016), and two Amazon co-purchase graphs, Computers and Photo (Shchur et al., 2018). We also use heterogeneous graphs, including Wikipedia graphs Chameleon and Squirrel (Rozemberczki et al., 2021), the Actor co-occurrence graph, and the webpage graph Texas and Cornell from WebKB3 (Pei et al., 2020). Their statistics are listed in Appendix G. We perform the node classification task, where we randomly split the node set into train/validation/test sets with a ratio of $60\%/20\%/20\%$. JacobiConv is compared with spectral GNNs: GCN, APPNP, ChebyNet, GPRGNN, and BernNet. Note that all these baselines use **nonlinear** transformations, while JacobiConv is a purely **linear** model. Results are shown in Table 2. Settings of these models are detailed in Appendix H.

JacobiConv outperforms all existing models on 9 out of 10 datasets and achieves performance gains up to 12% on a heterogeneous dataset Squirrel. On the Actor dataset, JacobiConv beats all baselines except BernNet. The generally top and runner-up performance of JacobiConv and BernNet verify our analysis in Section 5.3. The results indicate that JacobiConv is a general spectral GNN with consistently good performance across datasets. They also show that nonlinearity is not necessary for learning powerful spectral filters given a good choice of polynomial basis.

7.3. Ablation Analysis

To illustrate the effectiveness of Jacobi basis, we compare JacobiConv with linear GNNs with other filter bases in the left part of Table 3. We also remove PCD from the models to ensure fairness as the coefficient distribution of different bases varies. Jacobi basis outperforms any other basis by more than 0.8% on average. Bernstein basis also outperforms Monomial on average, which is consistent with the results in Table 2.

In the right part of Table 3, UniFilter is JacobiConv using the same filter for all prediction dimensions. No-PCD is JacobiConv without PCD. The results illustrate that the multiple filter functions, PCD, and the Jacobi basis are all

Table 2. Results on real-world datasets: Mean accuracy (%) ± 95% confidence interval.

DATASETS	GCN	APPNP	CHEBYNET	GPRGNN	BERNNET	JACOBICONV
CORA	$87.14_{\pm 1.01}$	$88.14_{\pm 0.73}$	$86.67_{\pm 0.82}$	$88.57_{\pm 0.69}$	$88.52_{\pm 0.95}$	$\mathbf{88.98}_{\pm \mathbf{0.46}}$
CITESEER	$79.86_{\pm 0.67}$	$80.47_{\pm 0.74}$	$79.11_{\pm 0.75}$	$80.12_{\pm 0.83}$	$80.09_{\pm 0.79}$	$\mathbf{80.78}_{\pm \mathbf{0.79}}$
PUBMED	$86.74_{\pm 0.27}$	$88.12_{\pm 0.31}$	$87.95_{\pm 0.28}$	$88.46_{\pm 0.33}$	$88.48_{\pm 0.41}$	$\mathbf{89.62}_{\pm \mathbf{0.41}}$
COMPUTERS	$83.32_{\pm 0.33}$	$85.32_{\pm 0.37}$	$87.54_{\pm 0.43}$	$86.85_{\pm 0.25}$	$87.64_{\pm 0.44}$	$\mathbf{90.39}_{\pm \mathbf{0.29}}$
PHOTO	$88.26_{\pm 0.73}$	$88.51_{\pm 0.31}$	$93.77_{\pm 0.32}$	$93.85_{\pm 0.28}$	$93.63_{\pm 0.35}$	$\mathbf{95.43}_{\pm \mathbf{0.23}}$
CHAMELEON	$59.61_{\pm 2.21}$	$51.84_{\pm 1.82}$	$59.28_{\pm 1.25}$	$67.28_{\pm 1.09}$	$68.29_{\pm 1.58}$	$\mathbf{74.20}_{\pm \mathbf{1.03}}$
ACTOR	$33.23_{\pm 1.16}$	$39.66_{\pm 0.55}$	$37.61_{\pm 0.89}$	$39.92_{\pm 0.67}$	$\mathbf{41.79}_{\pm \mathbf{1.01}}$	$41.17_{\pm 0.64}$
SQUIRREL	$46.78_{\pm 0.87}$	$34.71_{\pm 0.57}$	$40.55_{\pm 0.42}$	$50.15_{\pm 1.92}$	$51.35_{\pm 0.73}$	$\mathbf{57.38}_{\pm \mathbf{1.25}}$
TEXAS	$77.38_{\pm 3.28}$	$90.98_{\pm 1.64}$	$86.22_{\pm 2.45}$	$92.95_{\pm 1.31}$	$93.12_{\pm 0.65}$	$\mathbf{93.44}_{\pm \mathbf{2.13}}$
CORNELL	$65.90_{\pm 4.43}$	$91.81_{\pm 1.96}$	$83.93_{\pm 2.13}$	$91.37_{\pm 1.81}$	$92.13_{\pm 1.64}$	$\mathbf{92.95}_{\pm \mathbf{2.46}}$

Table 3. Results of ablation study on real-world datasets: Mean accuracy (%) ± 95% confidence interval.

DATASETS	MONOMIAL	CHEBYSHEV	BERNSTEIN	JACOBI	JACOBICONV	UNIFILTER	NO-PCD	NL-RES	NL
CORA	$88.80_{\pm 0.67}$	$88.49_{\pm 0.82}$	$86.50_{\pm 1.26}$	$\mathbf{88.98}_{\pm \mathbf{0.72}}$	$88.98_{\pm 0.46}$	$\mathbf{89.05}_{\pm \mathbf{0.48}}$	$88.98_{\pm 0.72}$	$89.00_{\pm 0.61}$	$88.67_{\pm 0.69}$
CITESEER	$\mathbf{80.68}_{\pm \mathbf{0.86}}$	$80.53_{\pm 0.81}$	$80.61_{\pm 0.85}$	$80.61_{\pm 0.72}$	$\mathbf{80.78}_{\pm \mathbf{0.79}}$	$80.42_{\pm 0.98}$	$80.61_{\pm 0.71}$	$80.16_{\pm 0.86}$	$80.25_{\pm 0.60}$
PUBMED	$89.54_{\pm 0.36}$	$89.52_{\pm 0.46}$	$88.42_{\pm 0.32}$	$\mathbf{89.70}_{\pm \mathbf{0.34}}$	$89.62_{\pm 0.41}$	$89.58_{\pm 0.25}$	$\mathbf{89.70}_{\pm \mathbf{0.34}}$	$86.44_{\pm 2.05}$	$87.73_{\pm 2.13}$
COMPUTERS	$89.06_{\pm 0.24}$	$89.16_{\pm 0.47}$	$87.09_{\pm 0.38}$	$\mathbf{89.22}_{\pm \mathbf{0.39}}$	$90.39_{\pm 0.29}$	$\mathbf{90.45}_{\pm \mathbf{0.34}}$	$89.22_{\pm 0.42}$	$87.45_{\pm 2.15}$	$86.85_{\pm 2.67}$
PHOTO	$95.33_{\pm 0.25}$	$95.45_{\pm 0.27}$	$94.59_{\pm 0.26}$	$\mathbf{95.53}_{\pm \mathbf{0.27}}$	$95.43_{\pm 0.23}$	$95.26_{\pm 0.31}$	$\mathbf{95.53}_{\pm \mathbf{0.19}}$	$94.16_{\pm 0.78}$	$85.65_{\pm 8.25}$
CHAMELEON	$65.95_{\pm 1.20}$	$\mathbf{74.09}_{\pm \mathbf{0.85}}$	$70.24_{\pm 1.05}$	$72.95_{\pm 0.83}$	$\mathbf{74.20}_{\pm \mathbf{1.03}}$	$73.76_{\pm 1.03}$	$72.95_{\pm 0.83}$	$72.63_{\pm 0.99}$	$72.56_{\pm 1.01}$
ACTOR	$40.31_{\pm 0.82}$	$40.61_{\pm 0.64}$	$40.42_{\pm 0.50}$	$\mathbf{40.70}_{\pm \mathbf{0.98}}$	$\mathbf{41.17}_{\pm \mathbf{0.64}}$	$40.01_{\pm 0.96}$	$40.70_{\pm 0.98}$	$37.80_{\pm 1.32}$	$37.56_{\pm 0.88}$
SQUIRREL	$37.93_{\pm 0.62}$	$\mathbf{56.71}_{\pm \mathbf{0.89}}$	$44.48_{\pm 0.89}$	$55.77_{\pm 0.55}$	$\mathbf{57.38}_{\pm \mathbf{1.25}}$	$54.11_{\pm 0.82}$	$55.77_{\pm 0.55}$	$48.66_{\pm 6.65}$	$43.73_{\pm 6.94}$
TEXAS	$91.64_{\pm 2.46}$	$88.36_{\pm 3.93}$	$89.34_{\pm 2.46}$	$\mathbf{92.79}_{\pm \mathbf{1.97}}$	$\mathbf{93.44}_{\pm \mathbf{2.13}}$	$90.82_{\pm 2.30}$	$92.79_{\pm 1.97}$	$89.84_{\pm 3.28}$	$89.34_{\pm 3.12}$
CORNELL	$91.31_{\pm 2.13}$	$88.03_{\pm 3.28}$	$\mathbf{92.46}_{\pm \mathbf{2.63}}$	$92.30_{\pm 2.79}$	$\mathbf{92.95}_{\pm \mathbf{2.46}}$	$92.62_{\pm 2.46}$	$92.30_{\pm 2.62}$	$89.67_{\pm 2.30}$	$87.54_{\pm 3.11}$

Table 4. Parameters/per-epoch time (ms)/total training time (s).

DATASETS	JACOBICONV	APPNP	BERNNET	GPRGNN
CORA	10K/6.4/3.1	92K/3.6/1.2	92K /11.6/3.1	92K/4.3/0.9
CITESEER	22K/6.3/3.0	237K/3.7/1.3	237K/11.8/3.4	237K/4.5/1.0
PUBMED	2K/6.6/4.9	32K/3.9/2.0	32K/11.1/4.9	32K/4.5/1.8
COMPUTERS	8K/7.3/4.8	50K/6.0/2.5	50K/29.3/8.6	50K/6.5/1.6
PHOTOS	6K/6.4/4.8	48K/5.8/2.8	48K/15.3/6.2	48K/4.5/1.3
CHAMELEON	12K/6.5/4.4	149K/3.9/0.8	149K/11.0/2.8	149K/4.4/1.0
ACTOR	5K/6.5/3.4	60K/3.8/0.8	60K/10.9/3.5	60K/4.3/0.9
SQUIRREL	11K/6.3/6.1	134K/4.3/0.9	134K/15.7/4.9	134K/4.3/2.1
TEXAS	9K/6.6/3.4	109K/3.8/0.8	109K/11.3/2.4	109K/4.3/1.0
CORNELL	9K/6.5/3.4	109K/3.8/0.8	109K/11.0/2.4	109K/4.4/0.9

essential for JacobiConv. On average, the multiple filter technique provides 1.3% performance gain, and the PCD technique provides 0.8% performance gain.

We design two variants to analyze how removing nonlinearity affects performance: NL and NL-Res. NL replaces the linear transformation in JacobiConv with a 2-layer ReLU MLP, whose first-layer output has the same dimension as the model output dimension. Compared with NL, NL-Res uses residual connection, which adds the output of the first linear layer to the output of the MLP. NL-Res outperforms NL by 2% on average, while NL leads to 6% performance loss compared with JacobiConv. These results illustrate that linear GNN is expressive enough, and nonlinear transformations can hardly promote the expressive power. The better performance of NL-Res over NL might also be due to its closer relationship to linear GNNs. On the other hand, the lower performance after adding nonlinearity may be attributed to overfitting caused by extra parameters.

7.4. Scalability

As shown in Table 4, compared with other baselines with comparable depth, our model, on average, only uses 10% parameters, as it only uses a linear layer to convert node features to the output shape, while other models use MLPs. JacobiConv also has a similar computational overhead to other baselines, though taking more time than APPNP and GPRGNN due to more complex bases. Theoretically, it still has the same time complexity $O(Kmd)$ as APPNP and GPRGNN, where K is the degree of the polynomial, m is the number of edges in the graph, and d is the number of node feature dimensions, while BernNet's time complexity is $O(K^2md)$.

8. Conclusion

In this paper, we analyze the expressive power of spectral GNNs. We prove that even without nonlinearity, spectral GNNs can be universal under mild conditions. We further analyze the optimization of spectral GNNs, which motivates the proposed JacobiConv, a novel spectral GNN using Jacobi basis. JacobiConv outperforms the previous state-of-the-art method BernNet by up to 12% on real-world datasets without using nonlinearity, which verifies our theory.

Acknowledgements

The authors greatly thank the actionable suggestions from the reviewers. Zhang is partly supported by the CCF-Baidu Open Fund (NO.2021PP15002000).

References

Abboud, R., Ceylan, İ. İ., Grohe, M., and Lukasiewicz, T. The surprising power of graph neural networks with random node initialization. In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, pp. 2112–2118, 2021.

Babai, L. and Kucera, L. Canonical labelling of graphs in linear average time. In 20th Annual Symposium on Foundations of Computer Science (sfcs 1979), pp. 39–46. IEEE, 1979.

Balcilar, M., Renton, G., Héroux, P., Gaüzère, B., Adam, S., and Honeine, P. Analyzing the expressive power of graph neural networks in a spectral perspective. In International Conference on Learning Representations, 2021.

Bianchi, F. M., Grattarola, D., Livi, L., and Alippi, C. Graph neural networks with convolutional ARMA filters. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.

Boyd, S. P. and Vandenberghe, L. Convex Optimization. Cambridge University Press, 2009.

Burden, R. and Faires, J. Numerical Analysis. Cengage Learning, 2005.

Chen, J., Ma, T., and Xiao, C. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In International Conference on Learning Representations, 2018.

Chen, L., Chen, Z., and Bruna, J. On graph neural networks versus graph-augmented mlps. In International Conference on Learning Representations, 2021.

Chen, M., Wei, Z., Ding, B., Li, Y., Yuan, Y., Du, X., and Wen, J.-R. Scalable graph neural networks via bidirectional propagation. In Advances in Neural Information Processing Systems, 2020a.

Chen, Z., Villar, S., Chen, L., and Bruna, J. On the equivalence between graph isomorphism testing and function approximation with gnns. In Advances in Neural Information Processing Systems, pp. 15868–15876, 2019.

Chen, Z., Chen, L., Villar, S., and Bruna, J. Can graph neural networks count substructures? In Advances in Neural Information Processing Systems, 2020b.

Chien, E., Peng, J., Li, P., and Milenkovic, O. Adaptive universal generalized pagerank graph neural network. In International Conference on Learning Representations, 2021.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems, pp. 3837–3845, 2016.

Feng, X. and Zhang, Z. The rank of a random matrix. Applied Mathematics and Computation, 185(1):689–694, 2007.

Fout, A., Byrd, J., Shariat, B., and Ben-Hur, A. Protein interface prediction using graph convolutional networks. In Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 6533–6542, 2017.

Garg, V. K., Jegelka, S., and Jaakkola, T. S. Generalization and representational limits of graph neural networks. In Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pp. 3419–3430, 2020.

He, M., Wei, Z., Huang, Z., and Xu, H. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. Advances in Neural Information Processing Systems, 2021.

Keriven, N. and Peyré, G. Universal invariant and equivariant graph neural networks. In Advances in Neural Information Processing Systems, pp. 7090–7099, 2019.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In International Conference on Learning Representations, 2019a.

Klicpera, J., Weiß enberger, S., and Günnemann, S. Diffusion improves graph learning. In Advances in Neural Information Processing Systems, 2019b.

Li, P., Wang, Y., Wang, H., and Leskovec, J. Distance encoding: Design provably more powerful neural networks for graph representation learning. Advances in Neural Information Processing Systems, 2020.

Loukas, A. What graph neural networks cannot learn: depth vs width. In International Conference on Learning Representations. OpenReview.net, 2020.

Marco, A. and Martinez, J.-J. Polynomial least squares fitting in the bernstein basis. Linear Algebra and its Applications, 433(7):1254–1264, 2010.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. In Advances in Neural Information Processing Systems, pp. 2153–2164, 2019a.

Maron, H., Fetaya, E., Segol, N., and Lipman, Y. On the universality of invariant networks. In Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pp. 4363–4371, 2019b.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In The Thirty-Third Conference on Artificial Intelligence, pp. 4602–4609, 2019.

Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web, 1999.

Pei, H., Wei, B., Chang, K. C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In International Conference on Learning Representations, 2020.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-scale attributed node embedding. J. Complex Networks, 2021.

Sato, R., Yamada, M., and Kashima, H. Approximation ratios of graph neural networks for combinatorial problems. In Advances in Neural Information Processing Systems, pp. 4083–4092, 2019.

Sato, R., Yamada, M., and Kashima, H. Random features strengthen graph neural networks. Proceedings of the 2021 SIAM International Conference on Data Mining (SDM), pp. 333–341, 2021.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. CoRR, abs/1811.05868, 2018.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Process. Mag., 30(3):83–98, 2013.

Srinivasan, B. and Ribeiro, B. On the equivalence between positional node embeddings and structural graph representations. In International Conference on Learning Representations, 2020.

Wang, H., Yin, H., Zhang, M., and Li, P. Equivariant and stable positional encoding for more powerful graph neural networks. arXiv preprint arXiv:2203.00199, 2022.

Weisfeiler, B. and Leman, A. The reduction of a graph to canonical form and the algebra which appears therein. NTI, Series, 2(9):12–16, 1968.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pp. 6861–6871. PMLR, 2019.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. IEEE Trans. Neural Networks Learn. Syst., 32(1):4–24, 2021.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In International Conference on Learning Representations, 2019.

Xu, K., Zhang, M., Jegelka, S., and Kawaguchi, K. Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. volume 139, pp. 11592–11602, 2021.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In Proceedings of the 33nd International Conference on Machine Learning, volume 48, pp. 40–48, 2016.

Yao, L., Mao, C., and Luo, Y. Graph convolutional networks for text classification. Proceedings of the AAAI conference on artificial intelligence, 33:7370–7377, 2019.

Zhang, M. and Li, P. Nested graph neural networks. Advances in Neural Information Processing Systems, 34: 15734–15747, 2021.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In Proceedings of the AAAI conference on artificial intelligence, volume 32, 2018.

Zhang, M., Li, P., Xia, Y., Wang, K., and Jin, L. Labeling trick: A theory of using graph neural networks for multi-node representation learning. Advances in Neural Information Processing Systems, 34:9061–9073, 2021.

Zhu, M., Wang, X., Shi, C., Ji, H., and Cui, P. Interpreting and unifying graph neural networks with an optimization framework. In The Web Conference, pp. 1215–1226, 2021.

A. Existing Models

Table 5. The filter form of spectral GNNs.

MODEL	g	HYPERPARAMS	LEARNABLE	PFME
SGC (Wu et al., 2019)	$(1-\lambda)^K$	α, K		×
APPNP (Klicpera et al., 2019a)	$\sum_{k=0}^{K} \frac{\alpha^k}{1-\alpha}(1-\lambda)^k$	α, K		×
GNN-LF (Zhu et al., 2021)	$\frac{1-(1-\mu)(1-\lambda)}{1-(2-\mu+\frac{1}{\alpha})(1-\lambda)}$	α, μ		×
GNN-HF (Zhu et al., 2021)	$\frac{1+\beta(1-\lambda)}{1-(1-\beta-\frac{1}{\alpha})(1-\lambda)}$	a, b		×
CHEBYNET (Defferrard et al., 2016)	$\sum_{k=0}^{K} \alpha_k \cos(k \arccos(1-\lambda))$	K	α_k, K	$\surd$
GPRGNN (Chien et al., 2021)	$\sum_{k=0}^{K} \alpha_k (1-\lambda)^k$	K	α_k	$\surd$
ARMA (Bianchi et al., 2021)	$\sum_{k=0}^{K} \frac{b_k}{1-a_k(1-\lambda)}$	K	a_k, b_k	$\surd$
BERNNET (He et al., 2021)	$\sum_{k=0}^{K} \alpha_k \binom{K}{k}(1-\frac{\lambda}{2})^{K-k}(\frac{\lambda}{2})^k$	K	α_k	$\surd$
JACOBICONV (OUR MODEL)	$\sum_{k=0}^{K} \alpha_k \sum_{s=0}^{k} \frac{(k+a)!(k+b)!(-\lambda)^{k-s}(2-\lambda)^s}{2^k s!(k+a-s)!(b+s)!(k-s)!}$	K, a, b	α_k	$\surd$

B. Proofs

B.1. Proof of Theorem 4.1

We restate Theorem 4.1 as follows.

Theorem B.1. *Assuming all rows of $\tilde{X}$ are not zero vector, and no eigenvalue of $\hat{L}$ has multiplicity larger than 1, for all $Z \in \mathbb{R}^{n\times 1}$, there exists a linear GNN to produce it.*

Proof. First, we prove that $W^* \in \mathbb{R}^d$ exists so that all elements of $\tilde{X}W^*$ are not zero.

Consider the i^{th} row of $\tilde{X}W$ equals 0. In other words, $\tilde{X}_i W = 0$. Let the solution space of W be V_i. As $\tilde{X}_i \neq 0$, V_i is a proper subspace of $\mathbb{R}^d$. Therefore, $\mathbb{R}^d - \bigcup_{i=1}^{n} V_i \neq \emptyset$. All vectors W in $\mathbb{R}^d - \bigcup_{i=1}^{n} V_i \neq \emptyset$ can meet the requirements,

Then we filter $\tilde{X}W^*$ to produce the output. For all one-dimension prediction $Z \in \mathbb{R}^n$, $\tilde{Z} = U^T Z \in \mathbb{R}^n$. If there exists a polynomial that $g^*(\lambda_i) = R_i$, where R is a vector whose i^{th} row $R_i = \frac{\tilde{Z}_i}{(\tilde{X}W)_i}$, for $i \in \{1, 2, ..., n\}$, linear GNNs can produce Z.

As λ_i are different from each other, consider an $n-1$ degree polynomial, $g(\lambda_i) = \sum_{k=0}^{n-1} \theta_k \lambda_i^k$. The coefficient θ_k of g^* is the solution of the linear system $B\Theta = R$, where $B \in \mathbb{R}^{n\times n}$ and $B_{ij} = \lambda_i^{j-1}$, $\Theta \in \mathbb{R}^n$ and $\Theta_k = \theta_{k-1}$, $R \in \mathbb{R}^n$, gives the coeffcient of g. As B^T is a Vandermonde matrix and becomes nonsingular if eigenvalues are different from each other, a solution always exists. Therefore, linear GNNs can give arbitrary one-dimensional prediction.

□

B.2. Proof of Proposition 4.2

Assuming an output $Z \in \mathbb{R}^{n\times k}, k > 1$, that linear GNNs can express it is equivalent to that the equation $Z = g(\hat{L})XW$ has solution polynomial g and matrix W. The equation is equivalent to $\tilde{Z} = g(\Lambda)\tilde{X}W$. Let $\tilde{X}_{s_i}, i = 1, 2, ..., \text{rank}(\tilde{X})$ be a maximal linearly independent subset of the set of row vectors in $\tilde{X}$. We prove that linear GNNs cannot produce the prediction described in the following lemma.

Lemma B.2. *Assuming that all the elements of the s_i^{th} row of $\tilde{Z}$ are the same scalar $\tilde{Z}_{s_i} \in \mathbb{R} - \{0\}$, $i = 1, 2, ..., n$ and there exists $\tilde{Z}_{ij_1} \neq \tilde{Z}_{ij_2}$, where $i \in \{1, 2, ..., n\} - \{s_i | i = 1, 2, ..., rank(X)\}, j_1, j_2 \in \{1, 2, ..., k\}, j_1 \neq j_2$, no linear GNN can produce $U\tilde{Z}$.*

Proof. $\tilde{X} = U^T X$, where U is an orthogonal matrix. Therefore, $\text{rank}(\tilde{X}) = rank(X) < n$.

Let $\mathbb{I}$ denote the set $\{s_i | i = 1, 2, ..., \text{rank}(X)\}$. As $\tilde{X}_{sI}$ forms a maximal linearly independent row vectors of $\tilde{X}$. Therefore, there exists $M \in \mathbb{R}^{n\times \text{rank(X)}}$, $\tilde{X} = M\tilde{X}_{\mathbb{I}}$. Only consider the rows in $\mathbb{I}$ of the equation.

$$\tilde{Z}_{\mathbb{I}} = g(\Lambda)_{\mathbb{I}\mathbb{I}}\tilde{X}_{\mathbb{I}}W. \tag{19}$$

As all elements in $\tilde{Z}_{\mathbb{I}} \neq 0$, all diagonal elements of $g(\Lambda)_{\mathbb{I}\mathbb{I}} \neq 0$. Therefore,

$$g(\Lambda)_{\mathbb{I}\mathbb{I}}^{-1}\tilde{Z}_{\mathbb{I}} = \tilde{X}_{\mathbb{I}}W. \tag{20}$$

Therefore, all column vectors of $\tilde{Z}$ should be equal, because

$$\tilde{Z} = g(\Lambda)\tilde{X}W = g(\Lambda)M\tilde{X}_{\mathbb{I}}W = (g(\Lambda)Mg(\Lambda)_{\mathbb{I}}^{-1})\tilde{Z}_{\mathbb{I}}. \tag{21}$$

As all column vectors of $\tilde{Z}_{\mathbb{I}}$ are equal, column vectors of $\tilde{Z}$ are all the same, while we assume that there exists $i \in \{1, 2, ..., n\} - \{s_i | i = 1, 2, ..., \text{rank}(X)\}, j_1, j_2 \in \{1, 2, ..., n\}, j_1 \neq j_2$ that $\tilde{Z}_{ij_1} \neq \tilde{Z}_{ij_2}$. Therefore, such linear GNNs do not exist.

□

B.3. Proof of Proposition 4.3 and Corollary 4.4

Proof. When the filter function is a K-degree polynomial, the prediction of the linear GNN can be formulated as follows.

$$Z = \sum_{k=0}^{K} \theta_k \hat{A}^k (XW). \tag{22}$$

Using the framework in (Xu et al., 2019), it can be considered as a $K+1$-layer GNN. Let $h_i^{(k)}$ denote the embeddings of node i at the k^{th} layer. COMBINE$^{(k)}$, AGGREGATE$^{(k)}$ are functions defined as follows.

$$\begin{aligned} a_i^{(1)} &= \text{AGGREGATE}^{(1)}(\{h_j^{(k-1)} | j \in N(i)\}) = |\{h_j^{(k-1)} | j \in N(i)\}| = D_{ii} \\ \text{COMBINE}^{(1)}(a_i^{(1)}, X_i) &= (D_{ii}, \theta_K X_i, X_i), \end{aligned} \tag{23}$$

where COMBINE$^{(1)}$ produce a tuple containing three items. For $k = 2, ..., K$,

$$\begin{aligned} a_i^{(k)} &= \text{AGGREGATE}^{(k)}(\{(D_{jj}, h_j^{(k-1)}, X_i) | j \in N(i)\}) = \sum_{j \in N(i)} \frac{1}{\sqrt{D_{jj}}} h_j^{(k-1)} \\ \text{COMBINE}^{(k)}(a^{(k)}, (D_{ii}, h_j^{(k-1)}, X_i)) &= (D_{ii}, \frac{1}{\sqrt{D_{ii}}} a_i^{(k)} + \theta_{K+1-k} X_i, X_i). \end{aligned} \tag{24}$$

For $k = K+1$,

$$\begin{aligned} a_i^{(k)} &= \text{AGGREGATE}^{(k)}(\{(D_{jj}, h_j^{(k-1)}, X_i) | j \in N(i)\}) = \sum_{j \in N(i)} \frac{1}{\sqrt{D_{jj}}} h_j^{(k-1)} \\ \text{COMBINE}^{(k)}(a^{(k)}, (D_{ii}, h_j^{(k-1)}, X_i)) &= \frac{1}{\sqrt{D_{ii}}} a_i^{(k)} + \theta_0 X_i. \end{aligned} \tag{25}$$

Therefore, the output of the last layer in GNN produce the output of linear GNNs. According to the proof of Lemma 2 in Xu et al. (2019), if WL node labels $WL_k(v) = WL_k(u)$, we always have GNN node features $h_i^{(k)} = h_j^{(k)}$ for any iteration i. Therefore, for all nodes $i, j \in \mathbb{V}$, $LG_K(i) = LG_K(j)$ if $WL_{K+1}(i) = WL_{K+1}(j)$.

The proof of Corollary 4.4 is obvious. For any pair of non-isomorphic nodes in the graph, linear GNNs can produce different outputs for the two nodes, so 1-WL can also differentiate them.

B.4. Proof of Theorem 4.5

Assuming π is a permutation function and P is a permutation matrix, $\delta_{\pi(a),a}$, the graph is isomorphic under the permutation π.

$$\begin{aligned} \hat{L} &= P^T \hat{L} P \\ U\Lambda U^T &= PU\Lambda U^T P^T \\ \Lambda &= U^T PU\Lambda U^T P^T U \\ \Lambda &= V\Lambda V^T, \end{aligned} \tag{26}$$

where V is an orthogonal matrix. As all diagonal elements of Λ are different, the eigenspace corresponding to each eigenvalue has only one dimension. Therefore,

$$U^T P U = V = D', \tag{27}$$

where D' is a diagonal matrix whose diagonal elements are ± 1. Therefore,

$$P = U D' U^T. \tag{28}$$

Therefore, P is symmetric, in other words, for $i \in \{1, 2, ..., n\}$, $\pi(\pi(i)) = i$. Therefore, for any graph without multiple normalized Laplacian eigenvalue, the order of permuatations is 1 or 2.

B.5. Proof of Theorem 4.6

For all permutation π and its matrix P for graph.

$$\begin{aligned} \hat{A} &= P\hat{A}P^T \\ X &= PX \end{aligned} \tag{29}$$

Let V denote $U^T P U$.

$$\begin{aligned} \Lambda &= V\Lambda V^T \\ \tilde{X} &= V\tilde{X} \end{aligned} \tag{30}$$

If $\hat{A}$ does not have multiple eigenvalues, $V = D$, D is a diagonal matrix whose diagonal elements are ± 1. So $(I - D)\tilde{X} = 0$.

Assuming all rows of$\tilde{X}$ are not zero vector (no missing frequency component), $I - D = 0$, $D = I$.

Therefore, $P = UDU^T = I$. Therefore, all pairs of nodes in this graph are not isomorphic when considering node features.

B.6. Proof of Proposition 5.1

Orthogonality require $\langle x, x\rangle \neq 0$ while $\langle 1, x^2\rangle = 0$. However,

$$\langle 1, x^2\rangle = \int_0^2 x^2 f(x)\mathrm{d}x = \langle x, x\rangle. \tag{31}$$

B.7. Proof of Proposition D.1

As $\tilde{X} = U^T X$, the distribution density f_1 of $\tilde{X}$ has a simple relation with the distribution density function f_2 of X,

$$\begin{aligned} f_1(\tilde{X}) &= f_2(U\tilde{X})|\det(U^T)| \\ &= \frac{1}{\det(2\pi\sigma^2 I)^{1/2}} e^{-\frac{1}{2}\tilde{X}^T U^T (\sigma^2 I)^{-1} U\tilde{X}} \\ &= \frac{1}{\det(2\pi\sigma^2 I)^{1/2}} e^{-\frac{1}{2}\tilde{X}^T (\sigma^2 I)^{-1}\tilde{X}} \end{aligned} \tag{32}$$

Therefore, $\tilde{X} \sim N_n(0, \sigma^2 I)$.

We can extend this proposition to the multi-dimensional cases. Consider $X \in \mathbb{R}^{n\times d}$, $\text{vec}(X) \in N_{nd}(0, \sigma^2 I)$, $\tilde{X} = U^T X$, $\text{vec}(\tilde{X}) = I \bigotimes U^T \text{vec}(X)$. $I \bigotimes U^T$ is still a orthogonal matrix. Therefore, $\text{vec}(\tilde{X}) \in N_{nd}(0, \sigma^2 I)$

B.8. Proof of Theorem D.2

We use a lemma from (Feng & Zhang, 2007).

Lemma B.3. *Let $F(x_1, ..., x_m)$ be a non-zero polynomial of variables $x_1, ..., x_m$ with real coefficients, then, $\mu_m D = 0$, where $D = \{x|F(x) = 0, x = (x_1, ..., x_m)^T \in \mathbb{R}^m\}$ and $\mu_m D$ is the Lebesgue measure of D as the set of points in $\mathbb{R}^m$.*

As we use individual filter parameters for each output dimension, if we can produce arbitrary one-dimensional prediction, muli-dimensional prediction can also be produced. So we can assume $Z \in \mathbb{R}^{n\times 1}$. Consider the linear GNNs in the frequency domain.

$$\tilde{Z} = g(\Lambda)\tilde{X}W. \tag{33}$$

Assuming that multiple eigenvalues are in the $i_1, i_2, ...$ rows of Λ. Let $\mathbb{I}$ be $i_1, i_2, ...$, $|\mathbb{I}| = \sum s_i$. As no frequency components are missing from Z, all diagonal elements in $g(\Lambda)$ are not zero.

We first build $g(\Lambda)_{\mathbb{I}\mathbb{I}}$ and $W_{\mathbb{I}}$ to produce $Z_{\mathbb{I}}$.

$$\tilde{Z}_{\mathbb{I}} = g(\Lambda)_{\mathbb{I}\mathbb{I}}\tilde{X}_{\mathbb{I}}W. \tag{34}$$

As all elements in $\tilde{X}$ independently follows $N(0, \sigma^2)$, the probability that $\tilde{X}_{\mathbb{I}}$ becomes singular is,

$$\int_{|\tilde{X}_{\mathbb{I}}|=0} \frac{1}{(2\pi\sigma^2)^{d^2/2}} e^{-\frac{1}{2\sigma^2}||X_{\mathbb{I}}||_F^2} \mathrm{d}\tilde{X}_{\mathbb{I}} \leq \int_{|\tilde{X}_{\mathbb{I}}|=0} \frac{1}{(2\pi\sigma^2)^{d^2/2}} \mathrm{d}\tilde{X}_{\mathbb{I}} = 0. \tag{35}$$

Therefore, $W = (\tilde{X}_{\mathbb{I}})^{-1} g(\Lambda)_{\mathbb{I}\mathbb{I}}^{-1} \tilde{Z}_{\mathbb{I}} \neq 0$. With probablity 1, $Z_{\mathbb{I}}$ can be produced.

Then we consider how to produce other rows. Let $\mathbb{J} = \{1, 2, ..., n\} - \mathbb{I}$.

$$\tilde{Z}_{\mathbb{J}} = g(\Lambda_{\mathbb{J}\mathbb{J}})\tilde{X}_{\mathbb{J}}W \tag{36}$$

The probability of some rows of $\tilde{X}_{\mathbb{J}}W$ are 0 is,

$$\begin{aligned} \int_{\min_{i\in\mathbb{J}} |\tilde{X}_{i'}W|=0} \frac{1}{(2\pi\sigma^2)^{d(n-d)/2}} e^{-\frac{1}{2\sigma^2}||X_{\mathbb{J}}||_F^2} \mathrm{d}\tilde{X}_{\mathbb{J}} &\leq \sum_{i'=1}^{n-1} \int_{|\tilde{X}_{i'}W|=0} \frac{1}{(2\pi\sigma^2)^{d(n-d)/2}} e^{-\frac{1}{2\sigma^2}||X_{\mathbb{J}}||_F^2} \mathrm{d}\tilde{X}_{\mathbb{J}} \\ &\leq \sum_{i'=1}^{n-1} \int_{|\tilde{X}_{i'}W|=0} \frac{1}{(2\pi\sigma^2)^{(n-d)d/2}} \mathrm{d}\tilde{X}_{\mathbb{J}} = 0. \end{aligned} \tag{37}$$

Assume that all rows of $\tilde{X}_{\mathbb{J}}W$ are not zero. As all elements of $\Lambda_{\mathbb{J}\mathbb{J}}$ are different, we can let $g(\Lambda)_{i_j} = \tilde{Z}_{i_j}/(\tilde{X}_{i_j}W)$. Therefore, other rows of $\tilde{Z}$ can also be built with probablity 1. The probability that Z can be produced is 1.

B.9. Proof of Proposition D.3

The number of different eigenvalues is $O(n)$. Let $\mathbb{I} = \{i_1, i_2, ...\}$ be the set of the index of different eigenvalues, and $\lambda_{i_1} < \lambda_{i_2} < ....$

Consider the signal $\tilde{x}$ in the frequency domain. $\tilde{x} \sim N(0, \sigma^2 I)$. For any pair of adjacent elements in $\tilde{x}_{\mathbb{I}}$, $\tilde{x}_{i_j}$ and $\tilde{x}_{i_{j+1}}$, the probability that two nodes have different signs is $\frac{1}{2}$. There, $O(n)$ pairs of $\tilde{x}_{i_j}$ and $\tilde{x}_{i_{j+1}}$ have different signs.

For these pairs, after filtering, $\tilde{z}_{i_j} = g(\lambda_{i_j})\tilde{x}_{i_j}$, $\tilde{z}_{i_{j+1}} = g(\lambda_{i_{j+1}})\tilde{x}_{i_{j+1}}$. There are three cases.

- $\tilde{z}_{i_j} = 0$ or $\tilde{z}_{i_{j+1}} = 0$. A zero-point exist for g.
- $\tilde{z}_{i_j}$ and $\tilde{z}_{i_{j+1}}$ have the same signs. A zero-point exist for g in $(\lambda_{i_j}, \lambda_{i_{j+1}})$.
- $\tilde{z}_{i_j}$ and $\tilde{z}_{i_{j+1}}$ have different signs.

Therefore, the number of zero points of g is the number of case 1 add that of case 2 minus the count of case 3, $O(n) - O(1) = O(n)$.

Therefore, the degree of polynomial is $O(n)$ in expectation.

C. Polynomial Filter with Limited Degree

Approximating functions with polynomials is well studied in numerical analysis. Weierstrass Approximation Theorem ensures the asymptotic approximation. For fixed-order polynomials, Theorem 3.3 of Burden & Faires (2005) shows that, when approximating a filter function $h \in C^{n+1}[0,2]$ with an n-order polynomial $g(x)$, an upper bound for the error exists.

$$sup_{x\in[0,2]}|h(x)-g(x)| \le \frac{1}{(n+1)!}(sup_{x\in[0,2]}|h^{(n+1)}(x)|)(sup_{x\in[0,2]}|\prod_{i=0}^{n}(x-x_i)|), \tag{38}$$

where $x_0, x_1, ..., x_n$ are distinct numbers selected in $[0,2]$. Let x_i be Chebyshev points $1+\cos(\frac{2i+1}{2n+2}\pi)$.

$$\begin{aligned} sup_{x\in[0,2]}|h(x)-g(x)| &\le \frac{1}{(n+1)!}(sup_{x\in[0,2]}|h^{(n+1)}(x)|)(sup_{x\in[0,2]}|\frac{1}{2^n}\cos((n+1)\arccos(x-1))|) &(39)\\ &\le \frac{1}{(n+1)!2^n}sup_{x\in[0,2]}|h^{(n+1)}(x)|. &(40)\end{aligned}$$

Therefore, the approximation error of polynomial depends on both the polynomial degree and the property of filter function. In linear GNNs, as each output dimension learns a different filter, we consider only one output dimension. The squared loss is bounded as follows.

$$\begin{aligned} \frac{1}{2}||Y-Z||_F^2 &= \frac{1}{2}(Z-Y)^T(Z-Y) &(41)\\ &= \frac{1}{2}(U\tilde{Y}-U\tilde{Z})^T(U\tilde{Y}-U\tilde{Z}) &(42)\\ &= \frac{1}{2}||\tilde{Y}-\tilde{Z}||_F^2 &(43)\\ &= \frac{1}{2}||(h(\Lambda)-g(\Lambda))\tilde{X}W||_F^2 &(44)\\ &\le \frac{1}{2}(\sup_{\lambda\in[0,2]}|h(\lambda)-g(\lambda)|)^2||\tilde{X}W||_F^2 &(45)\\ &\le \frac{1}{2}(\frac{1}{(n+1)!2^n})^2||XW||_F^2 sup_{x\in[0,2]}|h^{(n+1)}(x)|^2. &(46)\end{aligned}$$

D. Random Feature. Why? Why not?

Next, we study ways to break the no-missing-frequency condition in Theorem 4.1 to increase linear GNNs' expressive power.

Existing literature has tried to utilize random features for GNNs. GNN-RNI (Abboud et al., 2021) randomly initializes node embeddings and can approximate any functions mapping graphs to real numbers. Abboud et al. (2021) prove that GNN with random features can universally approximate any permutation invariant function $f: \mathcal{G}_n \to \mathbb{R}$, which mainly describes the representation of the whole graph. Sato et al. (2021) prove that GNN with random features can distinguish any local structure. Both works analyze from a graph isomorphism perspective. However, from a spectral perspective, we prove the expressive power of random features for node property tasks and analyze why it fails on node classification tasks.

First, we prove that no frequency component is missing from the random feature.

Proposition D.1. *Assume vector $x \sim N_n(0,\sigma^2 I)$, where N_n is the Gaussian distribution of n variables. The graph Fourier transformation of x is $\tilde{x} \sim N_n(0,\sigma^2 I)$.*

The proposition is proved in Appendix B.7.

We call x in Proposition D.1 random features. Therefore, the probability of some frequency components missing from the random features is 0. If we concatenate random features to the node features, no frequency components will be missing from the node features. Moreover, random features can also help with the multiple eigenvalue problem.

Table 6. Results on real-world datasets: Mean accuracy (%) ± 95% confidence interval.

Datasets	JacobiConv	Learnable	Random
Cora	$\mathbf{88.98_{\pm 0.46}}$	$82.82_{\pm 0.59}$	$23.46_{\pm 1.92}$
CiteSeer	$\mathbf{80.78_{\pm 0.79}}$	$71.17_{\pm 1.20}$	$18.99_{\pm 1.02}$
Pubmed	$\mathbf{89.62_{\pm 0.41}}$	$83.26_{\pm 0.46}$	$35.90_{\pm 0.83}$
Computers	$\mathbf{90.45_{\pm 0.34}}$	$83.69_{\pm 0.30}$	$18.40_{\pm 2.25}$
Photo	$\mathbf{95.43_{\pm 0.23}}$	$91.87_{\pm 0.19}$	$19.90_{\pm 2.71}$
Chameleon	$74.20_{\pm 1.03}$	$\mathbf{75.10_{\pm 0.83}}$	$25.03_{\pm 2.30}$
Actor	$\mathbf{41.17_{\pm 0.64}}$	$24.50_{\pm 1.10}$	$25.88_{\pm 0.43}$
Squirrel	$57.38_{\pm 1.25}$	$\mathbf{61.00_{\pm 1.09}}$	$20.26_{\pm 0.73}$
Texas	$\mathbf{93.44_{\pm 2.13}}$	$72.13_{\pm 5.41}$	$28.36_{\pm 5.57}$
Cornell	$\mathbf{92.95_{\pm 2.46}}$	$31.31_{\pm 11.48}$	$25.74_{\pm 7.21}$

Theorem D.2. *Assuming that the number of multiple eigenvalues is m, and among them, the $i-th$ multiple eigenvalue has multiplicity s_i. With $(\sum_{i=1}^{m} s_i)$-dimensional $\sim N_n(0, \sigma^2 I)$ random node features, for all prediction with no missing frequency components, linear GNNs can produce it with probability* 1.

The proof can be found in Appendix B.8.

We have seen the power of random features for improving the expressive power of linear GNNs. However, on large graphs, this technique can worsen the performance of models. As the coefficient of components of node features vibrates frequently, the filter function may be very complex even if we fit simple graph signals. Therefore, as formalized in Proposition D.3, $O(n)$-degree polynomial is needed, which is impossible to implement for large graphs.

Proposition D.3. *If $\hat{L}$ has no multiple eigenvalue, $O(n)$ degree polynomial is needed for linear GNN using Gaussian random features to predict a one-dimensional non-zero target whose coeffcients of frequency components can be expressed as a $O(1)$-degree polynomial.*

The proof of Proposition D.3 can be found in Appendix B.9.

$O(n)$-degree polynomial is too time- and memory-consuming for real-world datasets. In practice, we can only afford constant-degree polynomials (such as degree 10 in our experiments), which explains why random features usually worsen the performance.

To verify our analysis, we compare JacobiConv (our proposed model) with random features (Random), JacobiConv with learnable random features (Learnable), and the original JacobiConv in Table 6. Random features significantly worsen the performance, while learnable random features performs much better. Much to our surprise, Learnable even beats JacobiConv on two datasets, which indicates that node features may have little useful information in some datasets.

E. Can Bias Complete Missing Components?

Missing components hamper the expressive power of linear GNNs. Adding a bias to the linear transformation may alleviate this problem, as it can introduce new components. However, bias cannot solve this problem completely.

Proposition E.1. *There exists a graph of size n and $X \in \mathbb{R}^{n\times d}$ with missing components such that $\forall b \in \mathbb{R}^{1\times d'}, \forall W \in \mathbb{R}^{d\times d'}$, some frequency components are still missing from $XW + b$.*

Proof. Consider a graph $\mathcal{G}$ of size n whose $0, 1$ nodes are isolated. Let $\mathcal{S}_1$ denote the subgraph composed of the two isolated nodes. $\mathcal{S}_2$ means the subgraph composed of nodes $\{2, ..., n-1\}$. Let $\hat{L}$, $\hat{L}_{\mathcal{S}_1}$, $\hat{L}_{\mathcal{S}_2}$ denote the normalized Laplacian of $\mathcal{G}$,

$\mathcal{S}_1$, $\mathcal{S}_2$, respectively. We have

$$\hat{L}_{\mathcal{S}_1} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \tag{47}$$

$$\hat{L}_{\mathcal{S}_2} = U_{\mathcal{S}_2} \Lambda_{\mathcal{S}_2} U_{\mathcal{S}_2}^T \tag{48}$$

$$\hat{L} = \text{diag}(L_{\mathcal{S}_1}, L_{\mathcal{S}_2}) \tag{49}$$

$$\hat{L} = U_{\mathcal{G}} \Lambda U_{\mathcal{G}}^T, \tag{50}$$

where $U_{\mathcal{G}}^T = \text{diag}(\begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}, U_{\mathcal{S}_2}^T)$. Therefore, $\forall b$, the 0th row of $U_{\mathcal{G}}^T 1_n b$ is $\vec{0}$. Let $\tilde{X}_0 = 0$. Therefore, the 0th row of $U^T(XW + b)$ is $\vec{0}$. In other words, some components are missing from $XW + b$. □

F. Connection between Linear GNN and Polynomial Regression.

Let $\tilde{F}(x) = \frac{1}{F(2)} F(x)$. Take n' independent random variables $x_1, x_2, ..., x_n'$ from the $\tilde{F}$ distribution and set these variables as the points of linear regression. The element of the Gram matrix G' of the polynomial regression using $g_k, k = 0, 2, ..., K$ basis is,

$$G'_{k_1 k_2} = \sum_{i=1}^{n'} g_{k_1}(x_i) g_{k_2}(x_i). \tag{51}$$

By the weak law of large numbers of probability theory,

$$\lim_{n' \to \infty} \frac{G'_{k_1 k_2}}{n'} = \mathcal{E}\left[\frac{G'_{k_1 k_2}}{n'}\right] = \int_0^2 g_{k_1}(x) g_{k_2}(x) \tilde{f}(x) \mathrm{d}x. \tag{52}$$

Therefore, $n' \to \infty$, $\frac{G'_{ij}}{n} F(2) = G_{ij}$. Scalar $\frac{F(2)}{n}$ will not affect condition number. When $n \to \infty$, $\kappa(G')$, the condition number of Gram matrix of polynomial regression using bases $g_k, k = 0, 1, 2, .., K$ with points sampled from $\tilde{F}$ distribution, equals to, $\kappa(H)$, the condition number of linear GNNs' Hessian matrix. Therefore, we can use some conclusions on polynomial regression's bases.

G. Datasets

We summarize the statistics of these datasets in Table 7.

Table 7. Dataset statistics. N_{miss} is the number of missing frequency components. R_{multi} is the ratio of multiple eigenvalues in all different Laplacian eigenvalues (%).

DATASETS	CORA	CITESEER	PUBMED	COMPUTERS	PHOTO	CHAMELEON	SQUIRREL	ACTOR	TEXAS	CORNELL
$\lvert V \rvert$	2708	3327	19717	13752	7650	2277	5201	7600	183	183
$\lvert E \rvert$	5278	4552	44324	245861	119081	31371	198353	26659	279	277
N_{MISS}	0	0	0	0	0	0	0	0	0	0
R_{MULTI}	1.49	3.25	0.03	0.16	0.22	0.22	0.00	0.13	0.00	0.00

H. Experimental Settings

Computing infrastructure. We leverage Pytorch Geometric and Pytorch for model development. All experiments are conducted on an Nvidia A40 GPU on a Linux server.

Baselines. We directly use the results reported in (He et al., 2021). JacobiConv and linear GNN with other bases have fewer parameters than baselines, as linear GNN have a fixed number of parameters given the node feature dimension and output dimension.

Model hyperparameter for Synthetic Datasets. We use optuna to perform random searches. Hyperparameters were selected to minimize average loss on the fifty images. The best hyperparameters selected for each model can be found in our

Table 8. Average of sum of square loss over 50 images.

JacobiConv	**0.0003**	**0.0017**	**0.0253**	**0.0157**	**0.2972**
Monomial	2.3257	4.0960	10.9556	8.8068	10.8799
Chebyshev	1.0037	2.3468	7.8522	6.0297	9.1175
Bernstein	0.0110	0.0058	0.1517	0.1607	0.5705

code. For linear GNNs, we use different learning rate and weight decay for the linear layer W, parameters of PCD θ, and the linear combination parameters α. We select learning rate from $\{0.0005, 0.001, 0.005, 0.01, 0.05\}$, weight decay from $\{0.0, 5e-5, 1e-4, 5e-4, 1e-3\}$. We select PCD's γ from $\{0.5, 1.0, 1.5, 2.0\}$. Jacobi Basis' a and b are selected from $[-1.0, 2.0]$.

Model hyperparameter for real-world datasets. Hyperparameters were selected to optimize accuracy scores on the validation sets. We use different dropout for X and XW. Both dropout probabilities are selected from $[0.0, 0.9]$. Other parameters are searched in the same way as synthetic datasets.

Training process. We utilize Adam optimizer to optimize models and set an upper bound (1000) for the number of forward and backward processes. An early stop strategy is used, which finishes training if the validation score does not increase after 200 epochs for real-world datasets.

I. Synthetic Dataset Results of Models with PCD

Results are shown in Table 8. JacobiConv still outperforms all other bases. In general, there is little performance difference between models with PCD and those without. The reason for the invalidation of PCD can be that linear GNN trained with the squared loss on synthetic datasets can converge to a global minimum, and the effect of PCD to help convergence may be unimportant.

J. Analysis Using Gradient Flow

Using gradient flow method, i.e., gradient descent with infinitesimal steps (Xu et al., 2021), we analyze the optimization of linear GNN. We prove that linear GNN can converge to the global minimum of loss function under mild conditions.

Let $\mathbb{I}$ denote the training set containing nodes. The prediction of a linear GNN is,

$$Z_{il} = \sum_{k=0}^{K}\sum_{j=1}^{d} \alpha_{kl} (g(\hat{L})X)_{ij} W_{jl}. \tag{53}$$

Gradient flow method assumes that $\frac{\mathrm{d}}{\mathrm{d}t}W_{jl} = -\frac{\partial L}{\partial W_{jl}}$, $\frac{\mathrm{d}}{\mathrm{d}t}\alpha_{kl} = -\frac{\partial L}{\partial \alpha_{kl}}$.

Therefore,

$$\frac{\mathrm{d}}{\mathrm{d}t}L = \sum_{\text{all elements}} \frac{\mathrm{d}L}{\mathrm{d}Z_{\mathbb{II}}} \odot \frac{\mathrm{d}Z_{\mathbb{II}}}{\mathrm{d}t} = -\sum_{i\in\mathbb{I}}\sum_{l=1}^{d'} \frac{\partial L}{\partial Z_{il}} (\sum_{k=0}^{K} \frac{\partial Z_{il}}{\partial \alpha_{kl}} \frac{\partial L}{\partial \alpha_{kl}} + \sum_{j=1}^{d} \frac{\partial Z_{il}}{\partial W_{jl}} \frac{\partial L}{\partial W_{jl}}). \tag{54}$$

$$\tag{55}$$

$$\frac{\partial L}{\partial \alpha_{kl}} = \sum_{i\in\mathbb{I}} \frac{\partial L}{\partial Z_{il}} \frac{\partial Z_{il}}{\partial \alpha_{kl}}. \tag{56}$$

$$\frac{\partial L}{\partial W_{jl}} = \sum_{i\in\mathbb{I}} \frac{\partial L}{\partial Z_{il}} \frac{\partial Z_{il}}{\partial W_{jl}}. \tag{57}$$

Therefore,

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}L &= -\sum_{i\in\mathbb{I}}\sum_{l=1}^{d'}\frac{\partial L}{\partial Z_{il}}\Big(\sum_{k=0}^{K}\frac{\partial Z_{il}}{\partial \alpha_{kl}}\sum_{i'\in\mathbb{I}}\frac{\partial L}{\partial Z_{i'l}}\frac{\partial Z_{i'l}}{\partial \alpha_{kl}}+\sum_{j=1}^{d}\frac{\partial Z_{il}}{\partial W_{jl}}\sum_{i'\in\mathbb{I}}\frac{\partial L}{\partial Z_{i'l}}\frac{\partial Z_{i'l}}{\partial W_{jl}}\Big)\\ &= -\sum_{l=1}^{d'}\sum_{i\in\mathbb{I},i'\in\mathbb{I}}\frac{\partial L}{\partial Z_{il}}\frac{\partial L}{\partial Z_{i'l}}\Big(\sum_{k=0}^{K}\frac{\partial Z_{il}}{\partial \alpha_{kl}}\frac{\partial Z_{i'l}}{\partial \alpha_{kl}}+\sum_{j=1}^{d}\frac{\partial Z_{il}}{\partial W_{jl}}\frac{\partial Z_{i'l}}{\partial W_{jl}}\Big)\\ &= -\sum_{l=1}^{d'}\frac{\partial L}{\partial Z_{\mathbb{I}l}}^{T}(M^{(l)}+S^{(l)})\frac{\partial L}{\partial Z_{\mathbb{I}l}},\end{aligned} \tag{58}$$

where $M^{(l)}=\frac{\partial Z_{\mathbb{I}l}}{\partial \alpha_{:l}}\frac{\partial Z_{\mathbb{I}l}}{\partial \alpha_{:l}}^{T}$ and $S^{(l)}=\frac{\partial Z_{\mathbb{I}l}}{\partial W_{:l}}\frac{\partial Z_{\mathbb{I}l}}{\partial W_{:l}}^{T}$, where we define $\frac{\partial \vec{a}}{\partial \vec{b}}$, the derivative of vector $\vec{a}$ with respect to vector $\vec{b}$, is a matrix whose i,j element is $\frac{\partial \vec{a}_i}{\partial \vec{b}_j}$.

Both $M^{(l)}$ and $S^{(l)}$ are symmetric semi-definite matrix. Let σ_l denote the minimum eigenvalue of $M^{(l)}+S^{(l)}$,

$$\frac{\mathrm{d}}{\mathrm{d}t}L \le -\sum_{l=1}^{d}\frac{\partial L}{\partial Z_{\mathbb{I}l}}^{T}\sigma_l\frac{\partial L}{\partial Z_{\mathbb{I}l}}. \tag{59}$$

If L is squared loss, namely $\frac{1}{2}\sum_{l=1}^{d'}\sum_{n\in\mathbb{I}}||Z_{nl}-Y_{nl}||_F^2$, and $\sigma_l>0$, linear GNN can always converge to global minimum of loss function.

$$\frac{\mathrm{d}}{\mathrm{d}t}L \le -\sum_{l}\sigma_l(Z_{\mathbb{I}l}-Y_{\mathbb{I}l})^T(Z_{\mathbb{I}l}-Y_{\mathbb{I}l}) \le -2\sigma_{\min}L, \tag{60}$$

where $\sigma_{\min}=\min_l\sigma_l$. Let L^* denote the minimum loss. $L^*>0$ and $\frac{\mathrm{d}L^*}{\mathrm{d}t}=0$.

$$\frac{\mathrm{d}}{\mathrm{d}t}(L-L^*) \le -2\sigma_{\min}(L-L^*). \tag{61}$$

Assuming L_t denotes the loss at time t,

$$(L_t-L^*) \le e^{-2\sigma_{\min}t}(L_0-L^*). \tag{62}$$

Therefore, we have a prior guarantee of linear convergence to a global minimum for any graph with $\sigma_{\min}>0$. For any desired $\epsilon>0$, we have that $L_0-L^*<\epsilon$ for any T such that

$$T \ge \frac{1}{2\sigma_{\min}}\log\frac{L_0-L^*}{\epsilon}. \tag{63}$$

K. Bases Visualization

We visualize Monomial, Chebyshev, Bernstein and Jacobi Bases in Figure 6. Note that Jacobi Bases change with hyperparameters. We use the hyperparameters for our image dataset here.

Table 9. Comparison between FullCoef and JacobiConv.

	Cora	Citeseer	Pubmed	Computers	Photo
FullCoef	87.37$_{\pm 1.20}$	**80.95$_{\pm 1.06}$**	89.37$_{\pm 0.64}$	89.94$_{\pm 0.46}$	94.93$_{\pm 0.50}$
JacobiConv	**88.98$_{\pm 0.46}$**	80.78$_{\pm 0.79}$	**89.62$_{\pm 0.41}$**	**89.96$_{\pm 0.29}$**	**95.43$_{\pm 0.23}$**
	Chameleon	Actor	Squirrel	Texas	Cornell
FullCoef	64.79$_{\pm 1.42}$	38.99$_{\pm 1.37}$	49.61$_{\pm 1.22}$	92.30$_{\pm 4.52}$	91.48$_{\pm 5.62}$
JacobiConv	**74.20$_{\pm 1.03}$**	**41.17$_{\pm 0.64}$**	**57.38$_{\pm 1.25}$**	**93.44$_{\pm 2.13}$**	**92.95$_{\pm 2.46}$**

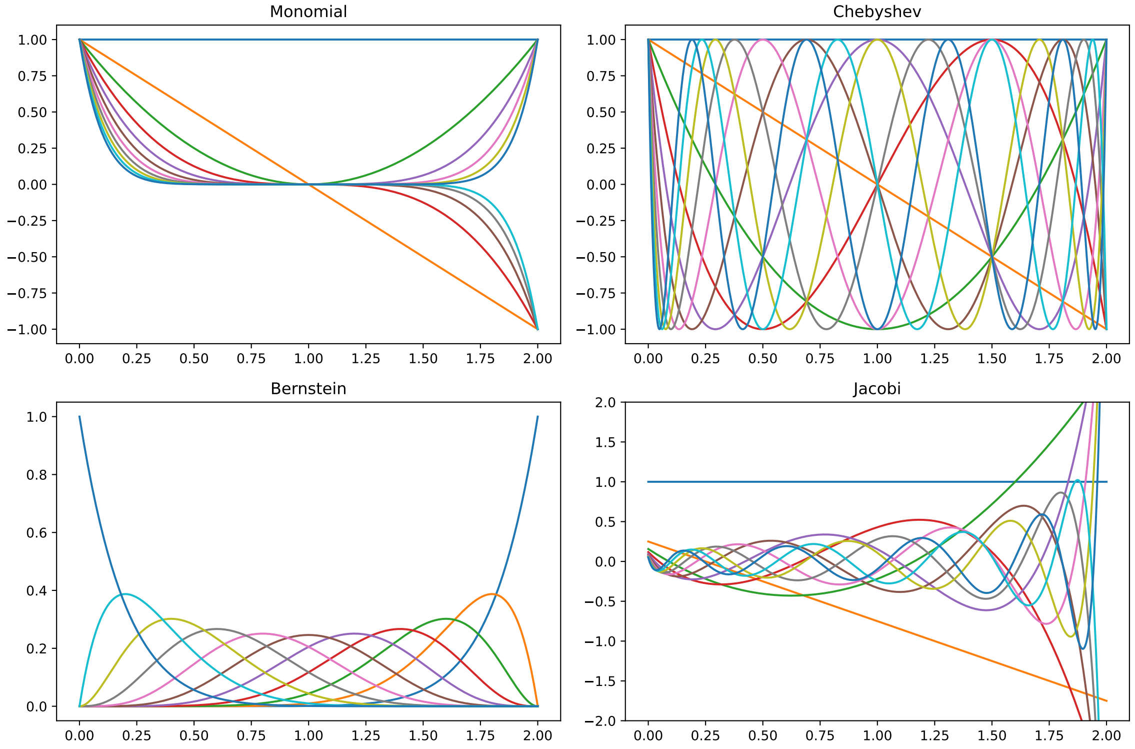

Figure 6. Polynomial bases visualization.

L. The Optimization of linear GNN with Different Polynomial Basis

In this section, we show how loss drops with different polynomial filter basis for linear GNNs in Figure 7.

On all five datasets, Jacobi basis achieves the lowest loss and a higher convergence rate than Monomial and Chebyshev basis. However, Bernstein polynomial basis shows a high optimization rate in a few first epochs, which may attribute to that the parameter is far from the local minimum initially, and our approximation fails. In contrast, after a few epochs, Jacobi basis approaches the local minimum and shows a higher convergence rate.

M. FullCoef vs JacobiConv

JacobiConv first linear transforms node features and then filters the signal in each output dimension individually. In contrast, some models like ChebyConv (Defferrard et al., 2016) use individual filter functions for each input dimension-output dimension pair to filter the signal in the input dimension and accumulate the filtered signal in the output dimension, which we call FullCoef. Though FullCoef may boost expressive power, extra parameters can also worsen the generalization. We compare FullCoef JacobiConv and original JacobiConv in Table 9. Results show that JacobiConv outperforms FullCoef on 9 out of 10 datasets. The extra express power that FullCoef brings is minor.

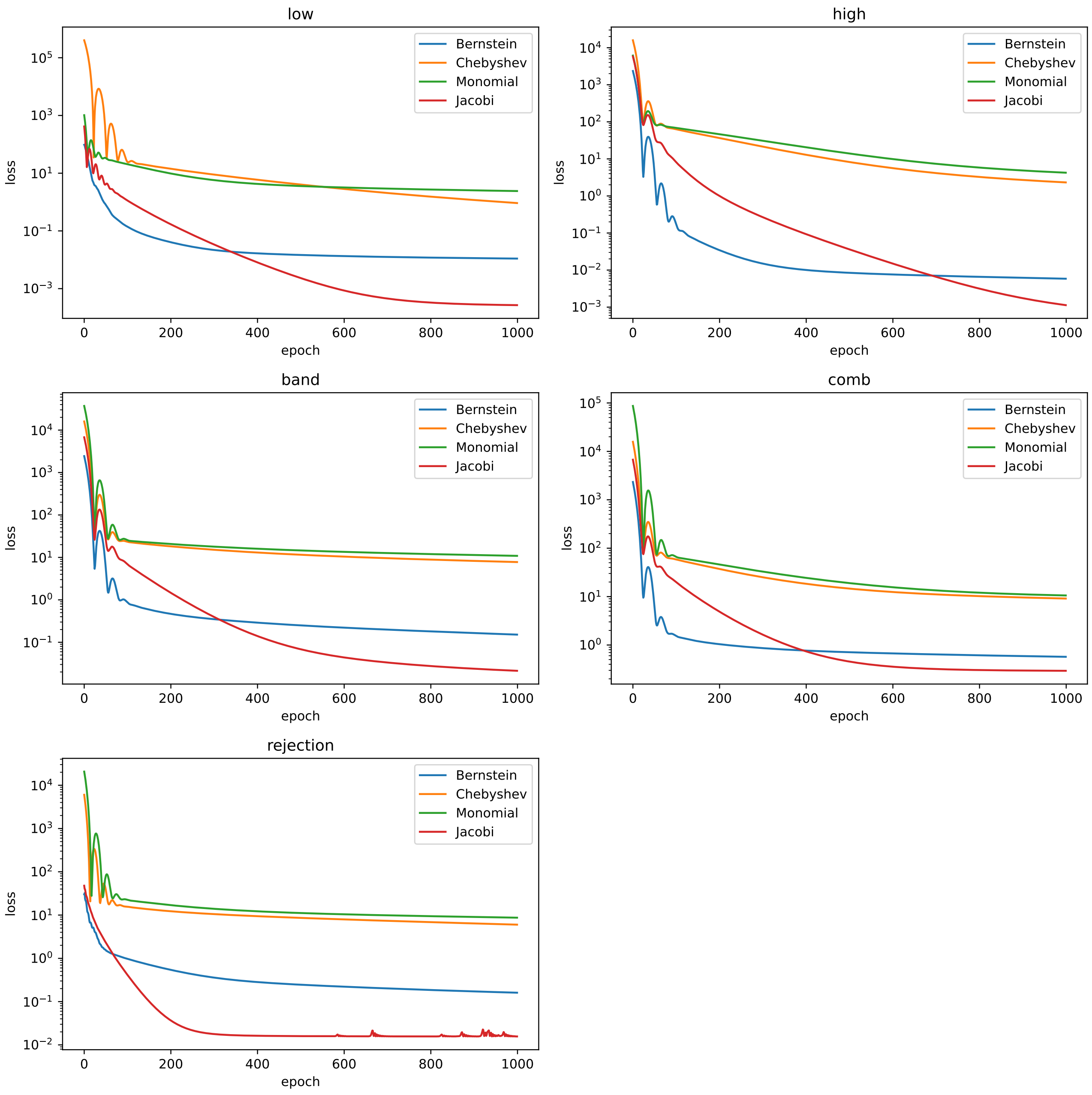

Figure 7. Training curve on some datasets.