CONVOLUTIONAL NEURAL NETWORKS ON GRAPHS WITH CHEBYSHEV APPROXIMATION, REVISITED

Mingguo He
Renmin University of China
mingguo@ruc.edu.cn

Zhewei Wei
Renmin University of China
zhewei@ruc.edu.cn

Ji-Rong Wen
Renmin University of China
jrwen@ruc.edu.cn

ABSTRACT

Designing spectral convolutional networks is a challenging problem in graph learning. ChebNet, one of the early attempts, approximates the spectral convolution using Chebyshev polynomials. GCN simplifies ChebNet by utilizing only the first two Chebyshev polynomials while still outperforming it on real-world datasets. GPR-GNN and BernNet demonstrate that the Monomial and Bernstein bases also outperform the Chebyshev basis in terms of learning the spectral convolution. Such conclusions are counter-intuitive in the field of approximation theory, where it is established that the Chebyshev polynomial achieves the optimum convergent rate for approximating a function.

In this paper, we revisit the problem of approximating the spectral convolution with Chebyshev polynomials. We show that ChebNet's inferior performance is primarily due to illegal coefficients learnt by ChebNet approximating **analytic** filter functions, which leads to over-fitting. We then propose ChebNetII, a new GNN model based on **Chebyshev interpolation**, which enhances the original Chebyshev polynomial approximation while reducing the Runge phenomena. We conducted an extensive experimental study to demonstrate that ChebNetII can learn arbitrary graph spectrum filters and achieve superior performance in both full- and semi-supervised node classification tasks.[1]

1 Introduction

Graph neural networks (GNNs) have received considerable attention in recent years due to their remarkable performance on a variety of graph learning tasks, including social analysis [27, 21, 33], drug discovery [43, 16, 28], traffic forecasting [22, 3, 7], recommendation system [36, 40] and computer vision [42, 5].

Spatial-based and spectral-based graph neural networks (GNNs) are the two primary categories of GNNs. To learn node representations, spatial-based GNNs [18, 13, 34] often rely on a message propagation and aggregation mechanism between neighboring nodes. Spectral-based methods [8] create spectrum graph convolutions or, equivalently, spectral graph filters, in the spectral domain of the Laplacian matrix. We can further divide spectral-based GNNs into two categories based on whether or not their graph convolutions can be learned.

- **Predetermined graph convolutions:** GCN [18] uses a simplified first-order Chebyshev polynomial as graph convolution, which is proven to be a low-pass filter [1, 35, 37, 44]. APPNP [19] utilizes Personalized PageRank (PPR) to set the graph convolution and achieves a low-pass filter as well [20, 44]. GNN-LF/HF [44] designs graph convolutions from the perspective of graph optimization functions, which can simulate high- and low-pass filters.
- **Learnable graph convolutions:** ChebNet [8] approximates the graph convolution with Chebyshev polynomials and learns the convolutional filters via trainable weights of the Chebyshev basis. GPR-GNN [6] uses the Monomial basis to approximate graph convolutions, which can derive high- or low-pass filters. ARMA [2] learns a rational convolutional filter via the family of Auto-Regressive Moving Average filters [24]. BernNet [15] utilizes the Bernstein basis to approximate graph convolutions, which can learn arbitrary graph spectral filters.

Despite the recent developments, two fundamental problems with spectral-based GNNs remain unsolved. First of all, it is well-known that GCN [18] outperforms ChebNet [8] on real-world datasets (e.g., semi-supervised node classification

[1] Preprint. Under review.

tasks on citation datasets [18]). However, it is also established that GCN is a simplified version of ChebNet with only the first two Chebyshev polynomials and that ChebNet has more expressive capability than GCN in theory [1]. Consequently, a natural question is: *Why is ChebNet's performance inferior to GCN's despite its better expressiveness?*.

Secondly, as shown in [15], the real-world performance of ChebNet is also inferior to that of GPR-GNN [6] and BernNet [15], which use Monomial polynomial basis and Bernstein polynomial basis to approximate the spectral convolution. Such a conclusion is counter-intuitive in the field of approximation theory, where it is established that the Chebyshev polynomial achieves near-optimum error when approximating a function. Therefore, the second question is: *Why is ChebNet's filter inferior to that of GPR-GNN and BernNet, despite the fact that Chebyshev polynomials have a higher approximation ability?*

In this paper, we attempt to tackle these problems by revisiting the fundamental problem of approximating the spectral graph convolutions with Chebyshev polynomials. First of all, according to the theory of the Chebyshev approximation, we observe that the coefficients of the Chebyshev expansion for an **analytic function** need to satisfy an inevitable convergence. Consequently, we prove that ChebNet's inferior performance is primarily due to illegal coefficients learnt by ChebNet approximating analytic filter functions, which leads to over-fitting. Furthermore, we propose ChebNetII, a new GNN model based on **Chebyshev interpolation**, which enhances the original Chebyshev polynomial approximation while reducing the Runge phenomena [9]. Our ChebNetII model can easily cope with various constraints on the learned filter via simple reparameterization, such as the non-negativity constraints proposed in [15]. Finally, we conduct an extensive experimental study to demonstrate that ChebNetII can learn arbitrary graph spectrum filters and achieve superior performance in both full- and semi-supervised node classification tasks.

2 Preliminaries

Notations. We consider an undirected graph $G = (V, E)$ with node set V and edge set E. Let $n = |V|$ denote the number of nodes. We use $\mathbf{x} \in R^n$ to denote the graph signal, where $\mathbf{x}(i)$ denote the signal at node i. Note that in the general case of GNN where the input feature is a matrix $\mathbf{X} \in R^{n\times f}$, we can treat each column of $\mathbf{X}$ as a graph signal and the result of this paper still applies. Let $\mathbf{A}$ denote the adjacency matrix and $\mathbf{D}$ denote the diagonal degree matrix of $\mathbf{A}$, where $\mathbf{D}_{ii} = \sum_j^n \mathbf{A}_{ij}$. For convenience, we use $\mathbf{P} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ and $\mathbf{L} = \mathbf{I} - \mathbf{P} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ to denote the symmetric normalized adjacency matrix and the symmetric normalized Laplacian matrix of G, respectively. We use $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ to denote the eigendecomposition of the symmetric normalized Laplacian matrix $\mathbf{L}$, where $\mathbf{U}$ is the matrix of eigenvectors and $\mathbf{\Lambda} = diag[\lambda_1, ..., \lambda_n]$ is the diagonal matrix of eigenvalues. Recent studies suggest that many popular GNNs operate as polynomial graph spectral filters [8, 18]. We can formulate its graph spectral filtering operation as

$$\mathbf{U}h\left(\Lambda\right)\mathbf{U}^T\mathbf{x} \approx \mathbf{U}\left(\sum_{k=0}^{K} w_k\mathbf{\Lambda}^k\right)\mathbf{U}^T\mathbf{x} = \left(\sum_{k=0}^{K} w_k\mathbf{L}^k\right)\mathbf{x}, \tag{1}$$

where w_k's denote the filter weights, and a polynomial graph filter can be denoted as $h(\lambda) = \sum_{k=0}^{K} w_k\lambda^k, \lambda \in [0, 2]$.

ChebNet. [8]: Defferrard et al. use Chebyshev polynomial to approximate a spectral filter for GNN. The filtering operation is defined as:

$$\mathbf{U}h\left(\Lambda\right)\mathbf{U}^T\mathbf{x} \approx \sum_{k=0}^{K} w_k T_k(\hat{\mathbf{L}})\mathbf{x}, \tag{2}$$

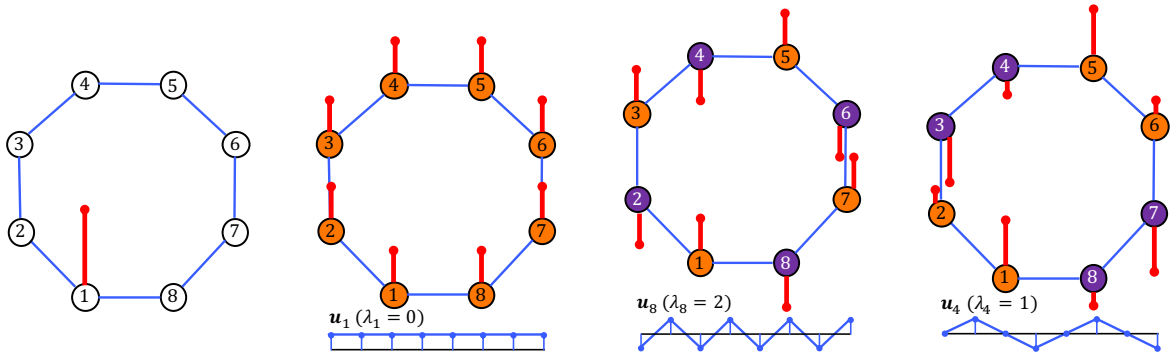
where $\hat{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I}$ denotes the scaled Laplacian matrix. λ_{max} is the largest eigenvalue of $\mathbf{L}$ and w_k is the Chebyshev coefficients. The Chebyshev polynomial are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. The model structure of ChebNet is defined as:

$$\mathbf{Y} = \sum_{k=0}^{K} T_k(\hat{\mathbf{L}})\mathbf{X}\mathbf{W}_k \tag{3}$$

where $\mathbf{W}_k$ is the trainable weights. The Chebyshev coefficients of filtering operation are implicitly encoded in the weight matrices $\mathbf{W}_k$.

Vanilla GCN. [18] The vanilla GCN simplifies the Chebyshev filter with the first-order approximation. In particular, the vanilla GCN sets $K = 1, w_0 = -w_1 = w$ and $\lambda_{max} = 2$ in equation (2) to obtain the filtering operation $\mathbf{U}h\left(\Lambda\right)\mathbf{U}^T\mathbf{x} = w(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}$. Finally, by the renormalization trick, the vanilla GCN replaces $\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ by a normalized version $\tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{A} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2}$. The graph convolution of vanilla GCN is defined as:

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\tilde{\mathbf{P}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}\right) \tag{4}$$



(a) Ring graph and the graph signal before filtering.

(b) Signal after impulse low-pass filter corresponds to homophilic labels.

(c) Signal after impulse high-pass filter corresponds to heterophilic labels.

(d) Signal after impulse band-pass filter corresponds to mixed homophilic or heterophilic labels.

Figure 1: A ring graph and filtering results. The red bar indicates the graph signal and the blue line charts represent the eigenvectors. Node labels are represented by orange and purple colors.

where σ denotes the activation function, $\mathbf{H}$ is the representation of each layer and $\mathbf{H}^{(0)} = \mathbf{X}$.

GPR-GNN. [6] GPR-GNN approximates graph spectral filters using the Monomial basis, which can also be understood as a Generalized PageRank. It has the following model structure:

$$\mathbf{Y} = \sum_{k=0}^{K} w_k \tilde{\mathbf{P}}^k f_\theta(\mathbf{X}) \tag{5}$$

where $f_\theta(\cdot)$ denotes a neural network with parameters $\{\theta\}$. Essentially, the graph spectral filter of GPR-GNN is $h(\tilde{\lambda}) = \sum_{k=0}^{K} w_k \tilde{\lambda}^k$, where $\tilde{\lambda}$ denotes the eigenvalues of $\tilde{\mathbf{P}}$.

BernNet. [15] BernNet utilizes the Bernstein basis to approximate graph spectral filters, whose model expression is defined as:

$$\mathbf{Y} = \sum_{k=0}^{K} w_k \frac{1}{2^K} \binom{K}{k} (2\mathbf{I} - \mathbf{L})^{K-k} \mathbf{L}^k f_\theta(\mathbf{X}) \tag{6}$$

where $f_\theta(\cdot)$ represents a neural network like GPR-GNN and the graph filter is $h(\lambda) = \sum_{k=0}^{K} w_k \frac{1}{2^K} \binom{K}{k} (2-\lambda)^{K-k} \lambda^k$, where $\frac{1}{2^K} \binom{K}{k} (2-\lambda)^{K-k} \lambda^k$ denotes the Bernstein basis.

3 Spectral Filter and Node Classification

In this section, we show why learning the right graph filter is crucial for node classification tasks on homophilic and heterophilic graphs. Given a graph signal vector $\mathbf{x}$, the filtering operation is defined as:

$$\mathbf{y} = \mathbf{U} diag\left[h(\lambda_1), ..., h(\lambda_n)\right] \mathbf{U}^T \mathbf{x} \tag{7}$$

where $\mathbf{y}$ denotes the filtering results of $\mathbf{x}$, and $h(\lambda)$ is called the graph spectral filter, which is a function of eigenvalues of the Laplacian matrix $\mathbf{L}$.

Figure 1 illustrates the relationship between various filters and the homophilic/heterophilic node labels on a ring graph. Figure 1(a) shows a ring graph, the red bar indicates the one-hot graph signal $\mathbf{x}$. When we apply low/high/band-pass filters on the graph signal $\mathbf{x}$, the output $\mathbf{y}$ can be used to classify homophilic/heterophilic node labels. For example, consider the homophilic graph 1(b) where all nodes have the same orange label. If we apply an impulse low-pass filter $h(\lambda) = \delta_0(\lambda)$ where $\delta_0(\lambda) = 1$ if $\lambda = 0$ and $\delta_0(\lambda) = 0$ for $\lambda > 0$, then the resulting graph signal $\mathbf{y}$ corresponds to the first eigenvector of the ring graph's Laplacian matrix. Consequently, the filtered signal evenly spreads over each node, which can be used to make perfect node classification on this homophilic graph.

On the other hand, figure 1(c) shows a heterophilic ring graph where any two connecting nodes have different labels (orange and purple). If we apply an impulse high-pass filter $h(\lambda) = \delta_2(\lambda)$ where $\delta_2(\lambda) = 1$ if $\lambda = 2$ and $\delta_2(\lambda) = 0$ for $\lambda < 2$, then the resulting graph signal $\mathbf{y}$ corresponds to the last eigenvector of the ring graph's laplacian matrix.

Table 1: The performance of ChebNet and GCN.

Method	Cora	Citeseer	Pubmed
ChebNet ($K = 2$)	$80.54_{\pm 0.38}$	$70.35_{\pm 0.33}$	$75.52_{\pm 0.75}$
ChebNet ($K = 10$)	$74.91_{\pm 0.52}$	$67.69_{\pm 0.64}$	$65.91_{\pm 1.71}$
GCN	$81.32_{\pm 0.18}$	$71.77_{\pm 0.21}$	$79.15_{\pm 0.18}$

Consequently, the filtered signal exhibits alternating signs, which can be used to make perfect node classification on this heterophilic graph. Finally, figure 1(d) shows a ring graph with mixed homophilic/heterophilic node labels, which can be handled by an impulse band-pass filter $h(\lambda) = \delta_1(\lambda)$ where $\delta_1(\lambda) = 1$ if $\lambda = 1$ and $\delta_1(\lambda) = 0$ otherwise.

The above example shows that the spectral filter is crucial to correct node classification on homophilic and heterophilic graphs. In fact, we have Theorem 3.1 (Appendix C shows the derivation.), stating that learning the right filter is all you need to make perfect node classification on any graphs.

Theorem 3.1. *For any graph $G = (V, E)$, if its node labels consist of two classes, we can always find a filter $h(\lambda)$ to to make perfect node classification.*

4 Revisiting ChebNet

4.1 The motivation of revisiting ChebNet

ChebNet vs. GCN. Even though GCN is a simplified form of ChebNet, it is well known that ChebNet is inferior to GCN for semi-supervised node classification tasks [18]. Table 1 shows the results of ChebNet and GCN for semi-supervised node classification tasks on Cora, Citeseer and Pubmed datasets. We found that ChebNet is outperformed by GCN, especially when we increase the polynomial order K from 2 to 10 in equation (3).

On the other hand, existing research [1] has shown that ChebNet is more expressive than GCN in theory. In particular, ChebNet can approximate arbitrary graph spectral filters as K increases, while GCN is a fixed low-pass filter. If we set $K = 1$ and $w_0 = w_1$ in the equation (2), ChebNet corresponds to a high-pass filter; if we set $K = 1$ and $w_0 = -w_1$, ChebNet becomes a low-pass filter which is essentially the same as GCN. Consequently, a natural question is: *Why is ChebNet's performance inferior to GCN's despite its better expressiveness?*.

Chebyshev basis vs. other bases. Chebyshev polynomials are widely used to approximate various functions in digital signal processing and graph signal filtering [31, 32]. It has been shown that for analytic functions in an ellipse containing the approximation interval, the truncated Chebyshev expansions give an approximate minimax polynomial [11]. Consequently, the graph spectral filter can be well-approximated by a truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to K-th order [14].

$$h(\hat{\lambda}) \approx \sum_{k=0}^{K} w_k T_k(\hat{\lambda}), \hat{\lambda} \in [-1, 1] \tag{8}$$

where $\hat{\lambda}_i \in [-1, 1]$ for $i = 1, 2, ..., n$ are the eigenvalues of the scaled Laplacian matrix $\hat{\mathbf{L}}$. ChebNet [8] then defined graph convolution using Chebyshev approximated filters, while the most recent work was inspired by ChebNet and used Monomial (i.e, GPR-GNN [6]) and Bernstein (i.e., BernNet [15]) bases to approximate graph filters. In order to evaluate the approximation ability of Chebyshev polynomials, we propose ChebNet with explicit coefficients, ChebBase, which simply replaces the Monomial basis of GPR-GNN and Bernstein basis of BernNet with a Chebyshev basis. The expression of ChebBase is defined as:

$$\mathbf{Y} = \sum_{k=0}^{K} w_k T_k(\hat{\mathbf{L}}) f_\theta(\mathbf{X}), \tag{9}$$

where $f_\theta(\mathbf{X})$ denotes Multi-Layer Perceptron (MLP). Table 2 reveals the results of ChebBase, GPR-GNN and BernNet for node classification tasks on three citation graphs. We can observe that ChebBase has the worst performance, which is inconsistent with the Chebyshev basis can approximate minimax polynomial in theory. Therefore, the second question is: *Why is ChebNet's filter inferior to that of GPR-GNN and BernNet, despite the fact that Chebyshev polynomials have a higher approximation ability?*

Table 2: The performance of different polynomial bases.

Method	Cora	Citeseer	Pubmed
ChebBase	$79.29_{\pm 0.36}$	$70.76_{\pm 0.37}$	$78.07_{\pm 0.32}$
GPR-GNN	$83.95_{\pm 0.22}$	$70.92_{\pm 0.57}$	$78.97_{\pm 0.27}$
BernNet	$83.15_{\pm 0.32}$	$72.24_{\pm 0.25}$	$79.65_{\pm 0.25}$

Table 3: Semi-supervised node classification on citation graphs

Method	Cora	Citeseer	Pubmed
ChebNet	$80.54_{\pm 0.38}$	$70.35_{\pm 0.33}$	$75.52_{\pm 0.75}$
GCN	$81.32_{\pm 0.18}$	$71.77_{\pm 0.21}$	$79.15_{\pm 0.18}$
ChebBase	$79.29_{\pm 0.36}$	$70.76_{\pm 0.37}$	$78.07_{\pm 0.32}$
ChebBase$/k$	$82.66_{\pm 0.28}$	$72.52_{\pm 0.29}$	$79.25_{\pm 0.31}$

4.2 Coefficient Constraints

We now demonstrate that ChebNet's suboptimal performance is due to the illegal coefficients learned, which result in over-fitting. Given an arbitrary continuous function $f(x)$ for $x \in [-1, 1]$, the Chebyshev expansion is defined as:

$$f(x) = \sum_{k=0}^{\infty} w_k T_k(x) \tag{10}$$

where w_k is the Chebyshev coefficients and T_k is the k-th Chebyshev polynomial. The following theorem establishes that in order to approximate an analytic function, the Chebyshev expansion's coefficients must be constrained.

Theorem 4.1. *[41] If $f(x)$ is weakly singular at the boundaries and analytic in the interval $(-1, 1)$, then the Chebyshev coefficients w_k will asymptotically (as $k \to \infty$) decrease proportionally to $1/k^q$ for some positive constant q.*

Here, "weakly singular" means that the derivative of f could vanish at the boundaries, and "analytic" means f can be locally given by a convergent power series in the interval $(-1, 1)$. Intuitively, since Chebyshev polynomial $T_k(x)$ with larger k corresponds to higher frequency oscillation (See Appendix A for more details). Theorem 4.1 essentially demonstrates that high frequency polynomials should be constrained in the Chebyshev expansion to approximate an analytic function.

The ability to approximate an analytic function is crucial in the task of approximating the graph convolution, since non-analytic filters are more difficult to approximate by polynomials and may result in over-fitting. In particular, ChebNet and ChebBase learn the coefficients w_k's by gradient descent without any constraints. The coefficients may not satisfy Theorem 4.1, leading to their poor performance.

To validate this conjuncture, we conducted an empirical analysis of ChebBase with difference coefficient constraints. Inspired by Theorem 4.1, we use the following propagation process for the ChebBase$/k$:

$$\mathbf{Y} = \sum_{k=0}^{K} \frac{w_k}{k} T_k(\hat{\mathbf{L}}) f_\theta(\mathbf{X}), \tag{11}$$

Table 3 shows the experimental results of the semi-supervised node classification performed on the citation graphs Cora, Citeseer, and Pubmed. We can observe that with a simple penalty on the coefficients w_k, ChebBase$/k$ outperforms ChebNet, ChebBase, and GCN, which validate Theorem 4.1.

5 ChebNetII model

Although ChebBase$/k$ appears to be a promising approach, it still has some drawbacks: 1) Imposing the penalty on the coefficients is not mathematically elegant, as Theorem 4.1 only provides a necessary condition for the coefficients; 2) It is hard to impose further constraints on the learned filters. For example, it is unclear how we can modify (11) to obtain non-negative filters, a requirement proposed in [15]. In this section, we describe ChebNetII, a GNN model based on Chebyshev polynomial interpolation that resolves the above two issues. We also discuss the advantages and disadvantages of various polynomial interpolations as well as the Runge phenomenon.

5.1 Chebyshev interpolation

Consider a real filter function $h(\hat{\lambda})$ that is continuous on the interval $\hat{\lambda} \in [-1, 1]$. When values of this filter are known at a finite number of point $\hat{\lambda}_k$, one can consider the approximation by a polynomial P_K with K degree such that $h(\hat{\lambda}_k) = P_K(\hat{\lambda}_k)$. Lemmas B.1 and B.2 in Appendix give an explicit expression for the lowest degree polynomial satisfying these interpolation conditions.

We generally sample the $K+1$ points $\hat{\lambda}_0 < \hat{\lambda}_1 < ... < \hat{\lambda}_K$ uniformly from $[-1, 1]$ to construct the interpolating polynomial $P_K(\hat{\lambda})$. Intuitively, increasing K should improve the approximation quality. However, this is not always the case due to the Runge Phenomenon [9] (The details are discussed in section 5.3). The popular approach to this problem in the literature [12] is Chebyshev interpolation, having superior approximation and faster convergence. Instead of sampling the interpolation points uniformly, Chebyshev interpolation uses Chebyshev nodes as interpolation points, which are essentially the zeros of the $(K+1)$-th Chebyshev polynomial.

Definition 5.1. ***(Chebyshev Nodes)*** *The Chebyshev polynomial $T_k(x)$ admits the closed form expression $T_k(x) = \cos(k \arccos(x))$. The Chebyshev Nodes for $T_k(x)$ are defined as $x_j = \cos\left(\frac{2j+1}{2k}\pi\right), j = 0, 1, ..., k-1$ which lie in the interval $(-1, 1)$ and are the zeros of $T_k(x)$.*

Definition 5.1 suggests that each Chebyshev polynomial $T_k(x)$ has k zeros, and we can define Chebyshev interpolation by replacing the equispaced points with Chebyshev nodes in polynomial interpolation (i.e., B.2). More eloquently, definition 5.2 efficiently defines the Chebyshev interpolation polynomials via their orthogonality properties.

Definition 5.2. ***(Chebyshev Interpolation)*** *[12] Given a continuous filter function $h(\hat{\lambda})$, let $x_j = cos\left(\frac{j+1/2}{K+1}\pi\right), j = 0, \ldots, K$ denotes the Chebyshev nodes for T_{K+1} and $h(x_j)$ denotes the function value at node x_j. The Chebyshev interpolation of $h(\hat{\lambda})$ is defined to be*

$$P_K(\hat{\lambda}) = \sum_{k=0}^{K} c'_k T_k(\hat{\lambda}), \tag{12}$$

where the prime indicates that the first term is to be halved, that is, $c'_0 = 1/2 * c_0$, $c'_1 = c_1, \ldots, c'_K = c_K$, and

$$c_k = \frac{2}{K+1} \sum_{j=0}^{K} h(x_j) T_k(x_j). \tag{13}$$

5.2 ChebNetII via Chebyshev Interpolation

Inspired by Chebyshev interpolation, we propose ChebNetII, a graph convolutional network that approximates an arbitrary spectral filter $h(\hat{\lambda})$ with an optimal convergence rate. ChebNetII simply reparameterizes the filter value $h(x_j)$ in Equation (13) as a learnable parameter γ_j, which allows the model to learn an arbitrary graph spectral filter via gradient descent. More precisely, the ChebNetII model can be formulated as

$$\mathbf{Y} = \frac{2}{K+1} \sum_{k=0}^{K} \sum_{j=0}^{K} \gamma_j T_k(x_j) T_k(\hat{\mathbf{L}}) f_\theta(\mathbf{X}) \tag{14}$$

where $x_j = cos\left(\left(j + 1/2\right)\pi/(K+1)\right)$ are the Chebyshev nodes of T_{K+1}, $f_\theta(\mathbf{X})$ denotes a Multi-Layer Perceptron (MLP) on the node feature matrix $\mathbf{X}$, and γ_j for $j = 0, 1, ..., K$ are the learnable parameters. Note that similar to APPNP [19], we decouple feature propagation and transformation.

Consequently, the filtering operation of ChebNetII can be expressed as

$$\mathbf{U} h\left(\Lambda\right) \mathbf{U}^T \mathbf{x} \approx \frac{2}{K+1} \sum_{k=0}^{K} \sum_{j=0}^{K} \gamma_j T_k(x_j) T_k(\hat{\mathbf{L}}) \mathbf{x} \tag{15}$$

It is easy to see that compared to the filter of original ChebNet (2), we only make one simple change: reparameterizing the coefficient w_k by $w_k = \frac{2}{K+1} \sum_{j=0}^{K} \gamma_j T_k(x_j)$. However, this simple modification allows us to have more control on shaping the resulting filter, as Chebyshev interpolation suggests that γ_j directly corresponds to the filter value $h(x_j)$ at the Chebyshev node x_j. Furthermore, Chebyshev interpolation also provides the GNN model with a number of useful mathematical properties.

5.3 Analysis of ChebNetII

ChebNetII has several advantages over existing GNN models due to the unique nature of Chebyshev interpolation. From the standpoint of polynomial approximation and computational complexity, we compare ChebNetII with current related approaches such as GPR-GNN [6] and BernNet [15].

Near-minimax approximation. First of all, we examine ChbeNetII's capabilities in terms of function approximation. Theorem 5.1 exhibits that ChebNetII provides an approximation that is close to the best polynomial approximation for a filter h.

Theorem 5.1. *[23] An approximation $P_K^*(x)$ for a function $f(x)$ is said to be near-best/minimax approximation with a relative distance ρ if*

$$||f(x) - P_K^*(x)|| \leq (1+\rho)||f(x) - P_B^*(x)|| \tag{16}$$

where ρ is the Lebesgue constant, $P_B^*(x)$ is a best approximation, and $||.||$ represents the uniform norm (i.e., $||g|| = \max_{x\in[-1,1]} |g(x)|$). Then, we have $\rho \sim 2^K$ as $K \to \infty$ for polynomial interpolation (Lemma B.1), and $\rho \sim log(K)$ as $K \to \infty$ for Chebyshev interpolation.

Convergence. In comparison to BernNet [15], which uses the Bernstein basis, ChebNetII has a faster convergence rate for approximating a filter function. In particular, we have the following Theorem:

Theorem 5.2. *[12, 25] Let $P_K(x)$ be the approximation polynomial for the function $f(x)$. Then the error is given as:*

$$||f(x) - P_K(x)|| \leq E(K).$$

if $P_K(x)$ is obtained by Bernstein approximation, then $E(K) \sim (1+(2K)^{-2})\omega(K^{-1/2})$; if $P_K(x)$ is obtained by Chebyshev Interpolation, then $E(K) \sim C\omega(K^{-1})logK$ with a constant C, where ω is the modulus of continuity.

Runge phenomenon. In comparison to GPR-GNN [6], which uses the Monomial basis, BernNet has the advantage of reducing the Runge phenomenon. In particular, when we utilize polynomial interpolation (Lemma B.1) to approximate a Runge filter $h(\hat{\lambda})$ with a high degree over a set of equispaced interpolation points, this will cause oscillation along the edges of an interval. Consequently, as the degree of the polynomial increases, the interpolation error increases. We analyze the error of polynomial interpolation:

$$R_K(\hat{\lambda}) = h(\hat{\lambda}) - P_K(\hat{\lambda}) = \frac{h^{K+1}(\zeta)}{(K+1)!}\pi_{K+1}(\hat{\lambda}) \tag{17}$$

where $\pi_{K+1}(\hat{\lambda}) = \prod_{k=0}^{K}(\hat{\lambda} - \hat{\lambda}_k)$ denotes the nodal polynomial and ζ is the value in $[-1,1]$. The bad Runge phenomenon is due to the values of the nodal polynomial $\pi_{K+1}(\hat{\lambda})$, which tends to present very strong oscillations near the endpoints of the interval. In particular, for high-degree polynomial interpolation at equidistant points, we have

$$\lim_{K\to\infty}\left(\max_{-1\leq\hat{\lambda}\leq 1}|h(\hat{\lambda}) - P_K(\hat{\lambda})|\right) = \infty. \tag{18}$$

On the other hand, we have the following Theorem that explains that Chebyshev nodes can minimize and quantify this error due to the nodal polynomial, meaning Chebyshev interpolation minimizes the problem of the Runge phenomenon.

Theorem 5.3. *[12] Consider the Chebyshev nodes $x_j = cos\left((j+1/2)\,\pi/(K+1)\right)$, $j = 0, 1, ..., K$. Then the monic polynomial $\hat{T}_{K+1}(x) = \prod_{k=0}^{K}(x - x_j)$ is the polynomial of degree $K+1$ with the smallest possible uniform norm, i.e., $||\hat{T}_{K+1}(x)|| = 2^{-K}$.*

Computational complexity. In comparison to BernNet [15], which has a time complexity quadratic to the order K, ChebNetII can be computed in time linear to K. In particular, we first calculate the ChebNetII coefficient c_k (Equation (13)), which takes $O(K)$ time, and then plug that coefficient into Equation (14) for propagation, which also takes $O(K)$ time, the same as that of ChebNet.

6 Experiments

In this section, we conduct experiments to evaluate the performance of ChebNetII against the state-of-the-art graph neural networks on a wide variety of open graph datasets.

Dataset and Experimental setup. We evaluate ChebNetII on several real-world heterophilic and homophilic graphs for the Semi- and Full-supervised node classification tasks. The datasets include three homophilic citation graphs: Cora,

Table 4: Dataset statistics

Dataset	Nodes	Edges	Features	Classes	$\mathcal{H}(G)$
Cora	2708	5278	1433	7	0.825
Citeseer	3327	4552	3703	6	0.718
Pubmed	19717	44324	500	5	0.792
Chameleon	2277	31371	2325	5	0.247
Squirrel	5201	198353	2089	5	0.217
Actor	7600	26659	932	5	0.215
Texas	183	279	1703	5	0.057
Cornell	183	277	1703	5	0.301

Table 5: Mean classification accuracy of semi-supervised node classification with random splits.

Method	Cham.	Squi.	Texas	Corn.	Actor	Cora	Cite.	Pubm.
MLP	$26.36_{\pm 2.85}$	$21.42_{\pm 1.50}$	$32.42_{\pm 9.91}$	$36.53_{\pm 7.92}$	$29.75_{\pm 0.95}$	$57.17_{\pm 1.34}$	$56.75_{\pm 1.55}$	$70.52_{\pm 2.01}$
GCN	$38.15_{\pm 3.77}$	$31.18_{\pm 0.93}$	$34.68_{\pm 9.07}$	$32.36_{\pm 8.55}$	$22.74_{\pm 2.37}$	$78.89_{\pm 1.37}$	$69.71_{\pm 1.32}$	$78.81_{\pm 0.84}$
ChebNet	$37.15_{\pm 1.49}$	$26.55_{\pm 0.46}$	$36.35_{\pm 8.90}$	$28.78_{\pm 4.85}$	$26.58_{\pm 1.92}$	$78.08_{\pm 0.86}$	$67.87_{\pm 1.49}$	$73.96_{\pm 1.68}$
ARMA	$37.42_{\pm 1.72}$	$24.15_{\pm 0.93}$	$39.65_{\pm 8.09}$	$28.90_{\pm 10.07}$	$27.02_{\pm 2.31}$	$79.14_{\pm 1.07}$	$69.35_{\pm 1.44}$	$78.31_{\pm 1.33}$
APPNP	$32.73_{\pm 2.31}$	$24.50_{\pm 0.89}$	$34.79_{\pm 10.11}$	$34.85_{\pm 9.71}$	$29.74_{\pm 1.04}$	$82.39_{\pm 0.68}$	$69.79_{\pm 0.92}$	$\mathbf{79.97_{\pm 1.58}}$
GPR-GNN	$33.03_{\pm 1.92}$	$24.36_{\pm 1.52}$	$33.98_{\pm 11.90}$	$38.95_{\pm 12.36}$	$28.58_{\pm 1.01}$	$82.37_{\pm 0.91}$	$69.22_{\pm 1.27}$	$79.28_{\pm 2.25}$
BernNet	$27.32_{\pm 4.04}$	$22.37_{\pm 0.98}$	$43.01_{\pm 7.45}$	$39.42_{\pm 9.59}$	$29.87_{\pm 0.78}$	$82.17_{\pm 0.86}$	$69.44_{\pm 0.97}$	$79.48_{\pm 1.47}$
ChebNetII	$\mathbf{43.42_{\pm 3.54}}$	$\mathbf{33.96_{\pm 1.22}}$	$\mathbf{46.58_{\pm 7.68}}$	$\mathbf{42.19_{\pm 11.61}}$	$\mathbf{30.18_{\pm 0.81}}$	$\mathbf{82.42_{\pm 0.64}}$	$\mathbf{69.89_{\pm 1.21}}$	$79.51_{\pm 1.03}$

Citeseer, and Pubmed [30, 39], as well as five heterophilic graphs: the Wikipedia graphs Chameleon and Squirrel [29], the Actor co-occurrence graph, and webpage graphs Texas and Cornell from WebKB[2] [26]. Following [26], we measure the level of homophily of nodes in a graph using $\mathcal{H}(G) = \frac{1}{n}\sum_{v\in V}\frac{|u:(u,v)\in E \wedge y_v = y_u|}{|u:(u,v)\in E|}$, where y_v denotes the label of node v. We summarize the dataset statistics in Table 4. All the experiments are conducted on a machine with an NVIDIA TITAN V GPU (12GB memory), Intel Xeon CPU (2.20 GHz), and 1TB of RAM.

6.1 Semi-supervised node classification with polynomial based methods

Setting and baselines. For the semi-supervised node classification task, we compare ChebNetII to 7 polynomial approximation filter methods, including MLP, GCN [18], ARMA [2], APPNP [19], ChebNet [8], GPR-GNN [6] and BernNet [15]. For dataset splitting, we employ both random and fixed splits and report the results on random splits. The results of fixed splits will be discussed in the supplementary materials. Specifically, we apply the standard random training/validation/testing split [39] on the three citation datasets (i.e., Cora, Citeseer, and Pubmed), with 20 nodes per class for training, 500 nodes for validation, and 1,000 nodes for testing. Since this standard split can not be used for very small graphs (e.g. Texas), we use sparse splitting [6] with the training/validation/test sets accounting for 2.5%/2.5%/95%, respectively, on the five homophilic datasets.

For ChebNetII, we use equation (14) as the propagation process and use the $ReLu$ function to reparametrize γ_j, maintaining the non-negativity of the filters [15]. Note that we add a linear layer after the propagation to improve the expressiveness for Chameleon and Squirrel datasets. We set the hidden units as 64, and $K = 5$ for Cora, $K = 10$ for the other datasets. We employ the Adam SGD optimizer [17] with an early stopping of 200 and a maximum of 1000 epochs to train ChebNetII. For baselines, we use the officially released code for GPR-GNN and BernNet, and use the Pytorch Geometric library implementations [10] for other models (i.e., MLP, GCN, APPNP, ARMA, and ChbeNet). More details of hyper-parameters and baselines' settings are listed in the supplementary materials.

Results. We utilize accuracy (the micro-F1 score) with a 95% confidence interval as the evaluation metric. Table 5 reports the relevant results on 10 random splits. Boldface letters indicate the best result for the given confidence interval, and underlinings denote the next best result. We first observe that ChebNet is inferior to GCN even on heterophilic graphs, which concurs with our theoretical analysis that the illegal coefficients learned by ChebNet lead to over-fitting. ChebNetII, on the other hand, outperforms other methods on all datasets excluding Pubmed, where it also achieves top-2 classification accuracy. This quality is due to the fact that the learnable parameters γ_j of ChebNetII directly correspond to the filter value $h(x_j)$ at the Chebyshev node x_j, effectively preventing it from learning an illegal filter.

[2] `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/`

Table 6: Mean classification accuracy of full-supervised node classification with random splits.

Method	Cham.	Squi.	Texas	Corn.	Actor	Cora	Cite.	Pubm.
MLP	$46.59_{\pm 1.84}$	$31.01_{\pm 1.18}$	$86.81_{\pm 2.24}$	$84.15_{\pm 3.05}$	$40.18_{\pm 0.55}$	$76.89_{\pm 0.97}$	$76.52_{\pm 0.89}$	$86.14_{\pm 0.25}$
GCN	$60.81_{\pm 2.95}$	$45.87_{\pm 0.88}$	$76.97_{\pm 3.97}$	$65.78_{\pm 4.16}$	$33.26_{\pm 1.15}$	$87.18_{\pm 1.12}$	$79.85_{\pm 0.78}$	$86.79_{\pm 0.31}$
ChebNet	$59.51_{\pm 1.25}$	$40.81_{\pm 0.42}$	$86.28_{\pm 2.62}$	$83.91_{\pm 2.17}$	$37.42_{\pm 0.58}$	$87.32_{\pm 0.92}$	$79.33_{\pm 0.57}$	$87.82_{\pm 0.24}$
ARMA	$60.21_{\pm 1.00}$	$36.27_{\pm 0.62}$	$83.97_{\pm 3.77}$	$85.62_{\pm 2.13}$	$37.67_{\pm 0.54}$	$87.13_{\pm 0.80}$	$80.04_{\pm 0.55}$	$86.93_{\pm 0.24}$
APPNP	$52.15_{\pm 1.79}$	$35.71_{\pm 0.78}$	$90.64_{\pm 1.70}$	$91.52_{\pm 1.81}$	$39.76_{\pm 0.49}$	$88.16_{\pm 0.74}$	$80.47_{\pm 0.73}$	$88.13_{\pm 0.33}$
GCNII	$63.44_{\pm 0.85}$	$41.96_{\pm 1.02}$	$80.46_{\pm 5.91}$	$84.26_{\pm 2.13}$	$36.89_{\pm 0.95}$	$88.46_{\pm 0.82}$	$79.97_{\pm 0.65}$	**$89.94_{\pm 0.31}$**
TWIRLS	$50.21_{\pm 2.97}$	$39.63_{\pm 1.02}$	$91.31_{\pm 3.36}$	$89.83_{\pm 2.29}$	$38.13_{\pm 0.81}$	$88.57_{\pm 0.91}$	$80.07_{\pm 0.94}$	$88.87_{\pm 0.43}$
GPR-GNN	$67.49_{\pm 1.38}$	$50.43_{\pm 1.89}$	$92.91_{\pm 1.32}$	$91.57_{\pm 1.96}$	$39.91_{\pm 0.62}$	$88.54_{\pm 0.67}$	$80.13_{\pm 0.84}$	$88.46_{\pm 0.31}$
BernNet	$68.53_{\pm 1.68}$	$51.39_{\pm 0.92}$	$92.62_{\pm 1.37}$	$92.13_{\pm 1.64}$	$41.71_{\pm 1.12}$	$88.51_{\pm 0.92}$	$80.08_{\pm 0.75}$	$88.51_{\pm 0.39}$
ChebNetII	**$71.37_{\pm 1.01}$**	**$57.72_{\pm 0.59}$**	**$93.28_{\pm 1.47}$**	**$92.30_{\pm 1.48}$**	**$41.75_{\pm 1.07}$**	**$88.71_{\pm 0.93}$**	**$80.53_{\pm 0.79}$**	$88.93_{\pm 0.29}$

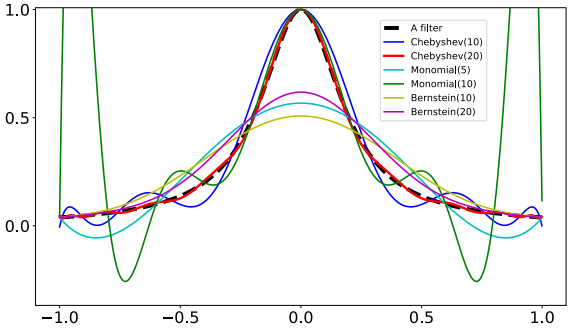


Figure 2: Approximate a filter via different polynomial bases.

6.2 Full-supervised node classification

Setting and baselines. For full-supervised node classification, we compare ChebNetII to the baselines in the prior semi-supervised node classification. We also include GCNII [4] and TWIRLS [38], two competitive baselines for full-supervised node classification. For all datasets, we randomly split the nodes into 60%, 20%, and 20% for training, validation and testing, and all methods share the same 10 random splits for a fair comparison, as suggested in [26, 15].

For ChebNetII, we set the hidden units to be 64 and $K = 10$ for all datasets, and employ the same training manner as in the semi-supervised node classification task. For GCNII and TWIRLS, we use the officially released code. More details of hyper-parameters and baselines' settings are listed in the supplementary materials.

Results. Table 6 reports the mean classification accuracy of each model. We first observe that, given more training data, ChebNet starts to outperform GCN on both homophilic and heterophilic datasets, which demonstrates the effectiveness of the Chebyshev approximation. However, we also observe that ChebNetII achieves new state-of-the-art results on 7 out of 8 datasets and top-2 results on all datasets. Notably, ChebNetII outperforms GPR-GNN and BernNet by over 10% on the Squirrel dataset. We attribute this quality to the fact that Chebyshev interpolation achieves near-minimax approximation of any function with respect to the uniform norm, giving ChebNetII greater approximation power than GPR-GNN and BernNet do.

6.3 Comparison of Different Polynomial Bases

We perform numerical studies comparing the Chebyshev basis to the Monomial and Bernstein bases to demonstrate ChebNetII's approximation power. Considering a Runge filter $h(\hat{\lambda}) = 1/(1 + 25\hat{\lambda}^2), \hat{\lambda} \in [-1, 1]$, figures 2 and 3 depict the approximation results and errors for several polynomial bases, with the polynomial degree K denoted by the numbers in brackets. We find that the Chebyshev basis has a faster convergence rate than the Bernstein basis and does not exhibit the Runge phenomenon compared to the Monomial basis. These findings provide empirical motivations for designing GNNs with Chebyshev interpolation.

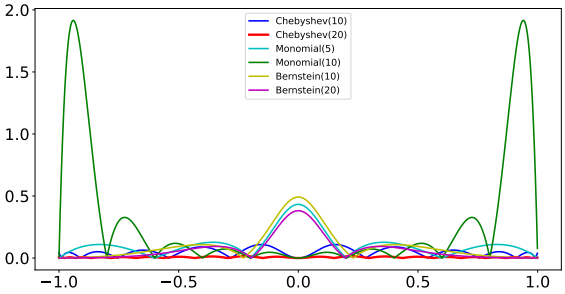


Figure 3: The error of approximating the filter via different polynomial bases.

7 Conclusion

This paper revisits the problem of approximating the spectral convolution with Chebyshev polynomials. We show that ChebNet's inferior performance is primarily due to illegal coefficients learned by approximating analytic filter functions, which leads to over-fitting. Moreover, we propose ChebNetII, a new GNN model based on Chebyshev interpolation, enhancing the original Chebyshev polynomial approximation while reducing the Runge phenomena. Experiments show that ChebNetII outperforms SOTA methods in terms of effectiveness on both real-world datasets.

References

[1] Muhammet Balcilar, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. Analyzing the expressive power of graph neural networks in a spectral perspective. In *ICLR*, 2021.

[2] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *TPAMI*, 2021.

[3] Toon Bogaerts, Antonio D Masegosa, Juan S Angarita-Zapata, Enrique Onieva, and Peter Hellinckx. A graph cnn-lstm neural network for short and long-term traffic forecasting based on trajectory data. *Transportation Research Part C: Emerging Technologies*, 2020.

[4] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, pages 1725–1735. PMLR, 2020.

[5] Zhao-Min Chen, Xiu-Shen Wei, Peng Wang, and Yanwen Guo. Multi-label image recognition with graph convolutional networks. In *CVPR*, 2019.

[6] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *ICLR*, 2021.

[7] Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *T-ITS*, 21(11):4883–4894, 2019.

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, pages 3844–3852, 2016.

[9] James F Epperson. On the runge example. *The American Mathematical Monthly*, 94(4):329–341, 1987.

[10] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. In *ICLR*, 2019.

[11] Keith O Geddes. Near-minimax polynomial approximation in an elliptical region. *SIAM Journal on Numerical Analysis*, 15(6):1225–1233, 1978.

[12] Amparo Gil, Javier Segura, and Nico M Temme. *Numerical methods for special functions*. SIAM, 2007.

[13] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[14] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[15] Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. In *NeurIPS*, 2021.

[16] Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun Hou. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of cheminformatics*, 13(1):1–23, 2021.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[18] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[19] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019.

[20] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019.

[21] Chang Li and Dan Goldwasser. Encoding social information with graph convolutional networks forpolitical perspective detection in news media. In *ACL*, 2019.

[22] Tanwi Mallick, Prasanna Balaprakash, Eric Rask, and Jane Macfarlane. Transfer learning with graph neural networks for short-term highway traffic forecasting. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 10367–10374. IEEE, 2021.

[23] John C Mason and David C Handscomb. *Chebyshev polynomials*. CRC press, 2002.

[24] Sunil K Narang, Akshay Gadde, and Antonio Ortega. Signal processing techniques for interpolation in graph structured data. In *ICASSP*. IEEE, 2013.

[25] Andrea Pallini. Bernstein-type approximations of smooth functions. *Statistica*, 65(2):169–191, 2005.

[26] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2020.

[27] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *KDD*, 2018.

[28] Prakash Chandra Rathi, R Frederick Ludlow, and Marcel L Verdonk. Practical high-quality electrostatic potential surfaces for drug discovery using a graph-convolutional deep neural network. *Journal of medicinal chemistry*, 63(16):8778–8790, 2019.

[29] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

[30] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[31] David I Shuman, Pierre Vandergheynst, and Pascal Frossard. Chebyshev polynomial approximation for distributed signal processing. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, pages 1–8. IEEE, 2011.

[32] Ljubisa Stankovic, Danilo Mandic, Milos Dakovic, Milos Brajovic, Bruno Scalzo, and Anthony G Constantinides. Graph signal processing–part ii: Processing and analyzing signals on graphs. *arXiv preprint arXiv:1909.10325*, 2019.

[33] Peihao Tong, Qifan Zhang, and Junjie Yao. Leveraging domain context for question answering over knowledge graph. *Data Science and Engineering*, 4(4):323–335, 2019.

[34] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[35] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

[36] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *AAAI*, 2019.

[37] Bingbing Xu, Huawei Shen, Qi Cao, Keting Cen, and Xueqi Cheng. Graph convolutional networks using heat kernel for semi-supervised learning. In *IJCAI*, 2019.

[38] Yongyi Yang, Tang Liu, Yangkun Wang, Jinjing Zhou, Quan Gan, Zhewei Wei, Zheng Zhang, Zengfeng Huang, and David Wipf. Graph neural networks inspired by classical iterative algorithms. In *ICML*, 2021.

[39] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, pages 40–48. PMLR, 2016.

[40] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018.

[41] Xiaolong Zhang and John P Boyd. Asymptotic coefficients and errors for chebyshev polynomial approximations with weak endpoint singularities: Effects of different bases. *arXiv preprint arXiv:2103.11841*, 2021.

[42] Long Zhao, Xi Peng, Yu Tian, Mubbasir Kapadia, and Dimitris N Metaxas. Semantic graph convolutional networks for 3d human pose regression. In *CVPR*, 2019.

[43] Tianyi Zhao, Yang Hu, Linda R Valsdottir, Tianyi Zang, and Jiajie Peng. Identifying drug–target interactions based on graph convolutional network and deep neural network. *Briefings in bioinformatics*, 22(2):2141–2150, 2021.

[44] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. Interpreting and unifying graph neural networks with an optimization framework. In *WWW*, 2021.

A Chebyshev basis

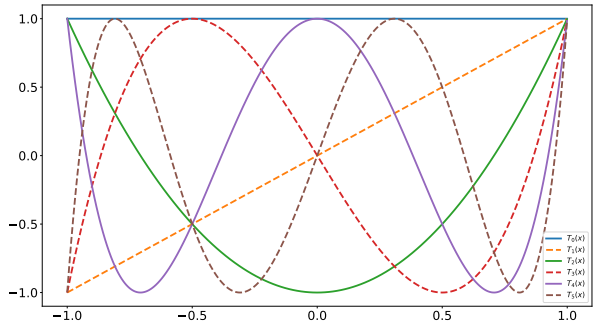


Figure 4: Chebyshev polynomials $T_k(x), k = 0, 1, 2, 3, 4, 5$.

The Chebyshev polynomials $T_k(x)$ for $x \in [-1, 1]$ satisfy the following three-term recurrence relation:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), k = 2, 3, \cdots$$

with starting values $T_0(x) = 1, T_1(x) = x$. The figure 4 exhibits the first six Chebyshev polynomials, and we can observe that Chebyshev polynomial $T_k(x)$ with larger k corresponds to higher frequency oscillation.

B More details for Polynomial Interpolation

Lemma B.1. *[23] Given a continuous function h that is defined on $[-1, 1]$, and given its values at $K + 1$ points $\hat{\lambda}_0 < \hat{\lambda}_1 < ... < \hat{\lambda}_K \in [-1, 1]$, there exists a unique polynomial of degree smaller than or equal to K such that*

$$P_K(\hat{\lambda}_k) = h(\hat{\lambda}_k), k = 0, 1, ..., K. \tag{19}$$

This polynomial is given by

$$P_K(\hat{\lambda}) = \sum_{k=0}^{K} a_k \hat{\lambda}^k \tag{20}$$

where a_k denotes the polynomial coefficient and can be uniquely derived by solving a Vandermonde linear system induced by equations (19) and (20). In particular, the Vandermonde linear system is defined as,

$$\begin{bmatrix} 1 & \hat{\lambda}_0 & \hat{\lambda}_0^2 & \cdots & \hat{\lambda}_0^K \\ 1 & \hat{\lambda}_1 & \hat{\lambda}_1^2 & \cdots & \hat{\lambda}_1^K \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \hat{\lambda}_K & \hat{\lambda}_K^2 & \cdots & \hat{\lambda}_K^K \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_K \end{bmatrix} = \begin{bmatrix} h(\hat{\lambda}_0) \\ h(\hat{\lambda}_1) \\ \vdots \\ h(\hat{\lambda}_K) \end{bmatrix} \tag{21}$$

Polynomial interpolation using the Vandermonde matrix is not practical because, for a large number of data points, it is expensive to solve the system of equations, and the matrix has a large condition number. As a result, Lagrange interpolation is an attractive and essentially equivalent approach to the polynomial interpolation using the Vandermonde matrix.

Lemma B.2. ***(Lagrange interpolation)*** *[23] Given a continuous function h that is defined on $[-1, 1]$, and given its values at $K + 1$ points $\hat{\lambda}_0 < \hat{\lambda}_1 < ... < \hat{\lambda}_K \in [-1, 1]$, there exists a unique polynomial of degree smaller than or equal to K such that*

$$P_K(\hat{\lambda}_k) = h(\hat{\lambda}_k), k = 0, 1, ..., K. \tag{22}$$

This polynomial is given by

$$P_K(\hat{\lambda}) = \sum_{k=0}^{K} h(\hat{\lambda}_k) L_k(\hat{\lambda}) \tag{23}$$

Table 7: Mean classification accuracy of semi-supervised node classification with fixed splits.

Method	Cham.	Squi.	Texas	Corn.	Actor	Cora	Cite.	Pubm.
MLP	$21.91_{\pm 2.11}$	$23.42_{\pm 0.94}$	$45.03_{\pm 2.45}$	$46.18_{\pm 5.10}$	$29.16_{\pm 0.52}$	$58.88_{\pm 0.62}$	$56.97_{\pm 0.54}$	$73.15_{\pm 0.28}$
GCN	$39.14_{\pm 0.60}$	$30.06_{\pm 0.75}$	$32.42_{\pm 2.23}$	$35.57_{\pm 3.55}$	$21.96_{\pm 0.54}$	$81.32_{\pm 0.18}$	$71.77_{\pm 0.21}$	$79.15_{\pm 0.18}$
ChebNet	$31.27_{\pm 1.34}$	$28.37_{\pm 0.28}$	$28.55_{\pm 3.28}$	$25.54_{\pm 3.42}$	$26.13_{\pm 1.27}$	$80.54_{\pm 0.38}$	$70.35_{\pm 0.33}$	$75.52_{\pm 0.75}$
ARMA	$40.58_{\pm 0.67}$	$28.45_{\pm 0.55}$	$47.84_{\pm 3.35}$	$30.89_{\pm 4.23}$	$27.26_{\pm 0.32}$	$83.15_{\pm 0.54}$	$71.41_{\pm 0.36}$	$78.75_{\pm 0.14}$
APPNP	$30.06_{\pm 0.96}$	$25.18_{\pm 0.35}$	$46.31_{\pm 3.01}$	$45.73_{\pm 4.85}$	$28.19_{\pm 0.31}$	$83.52_{\pm 0.24}$	$72.09_{\pm 0.25}$	$80.23_{\pm 0.15}$
GPR-GNN	$30.56_{\pm 0.94}$	$25.11_{\pm 0.51}$	$45.76_{\pm 3.78}$	$43.42_{\pm 4.95}$	$27.32_{\pm 0.83}$	$\mathbf{83.95_{\pm 0.22}}$	$70.92_{\pm 0.57}$	$78.97_{\pm 0.27}$
BernNet	$26.35_{\pm 1.04}$	$24.57_{\pm 0.72}$	$48.21_{\pm 3.17}$	$46.64_{\pm 5.62}$	$29.27_{\pm 0.23}$	$83.15_{\pm 0.32}$	$72.24_{\pm 0.25}$	$79.65_{\pm 0.25}$
ChebNetII	$\mathbf{46.45_{\pm 0.53}}$	$\mathbf{36.18_{\pm 0.46}}$	$\mathbf{54.68_{\pm 3.87}}$	$\mathbf{50.92_{\pm 5.49}}$	$\mathbf{29.54_{\pm 0.46}}$	$83.67_{\pm 0.33}$	$\mathbf{72.75_{\pm 0.16}}$	$\mathbf{80.48_{\pm 0.23}}$

where $L_i(\hat{\lambda})$ is defined by

$$L_k(\hat{\lambda}) = \frac{\pi_{K+1}(\hat{\lambda})}{(\hat{\lambda} - \hat{\lambda}_k)\pi'_{K+1}(\hat{\lambda})} = \frac{\prod_{j=0, j\neq k}^{K}(\hat{\lambda} - \hat{\lambda}_j)}{\prod_{j=0, j\neq k}^{K}(\hat{\lambda}_k - \hat{\lambda}_j)} \tag{24}$$

where $\pi_{K+1}(\hat{\lambda}) = \prod_{j=0}^{K}(\hat{\lambda} - \hat{\lambda}_j)$ denotes the nodal polynomial. If h is $K+1$ times differentiable in $[-1, 1]$, then for any $\hat{\lambda} \in [-1, 1]$ there exists a value $\zeta \in [-1, 1]$ (depending on $\hat{\lambda}$), such that

$$R_K(\hat{\lambda}) = h(\hat{\lambda}) - P_K(\hat{\lambda}) = \frac{h^{K+1}(\zeta)}{(K+1)!}\pi_{K+1}(\hat{\lambda}) \tag{25}$$

C Additional proofs

C.1 Theorem 3.1

For any graph $G = (V, E)$, if its node labels consist of two classes, we can always find a filter $h(\lambda)$ to to make perfect node classification.

Proof. For a graph with two classes of node labels, its equivalent is the label vector $\mathbf{y} \in R^n$ containing only two kinds of values. We assume $y_i \in \{1, -1\}$ for $i = 1, 2, \cdots, n$. Then, for a graph signal $\mathbf{x}$, we need to find a filter $h(\lambda)$ satisfying the below equation,

$$\mathbf{y} = \mathbf{U}diag[h(\lambda_1), ..., h(\lambda_n)]\mathbf{U}^T\mathbf{x}$$

We can observe that by making the element of $\mathbf{U}^T\mathbf{x}$ non-zero, we can always obtain n variables $(h(\lambda_1), \cdots, h(\lambda_n))$ to make the equation hold. If $\mathbf{x}$ and eigenvectors $u_i, i \in \{1, 2, \cdots, n\}$ are not orthogonal, then the element of $\mathbf{U}^T\mathbf{x}$ is not zero. Therefore, we can definitely get a filter $h(\lambda)$ to achieve perfect node classification, when we set $\mathbf{x}$ and eigenvectors u_i not orthogonal. Note that if $\mathbf{x}$ is orthogonal to a eigenvector u_i, then its expressiveness in the subspace for remaining eigenvectors must be confined.

□

D Additional experimental details

Baselines Implementations. For MLP, GCN, ChebNet, AMAR and APPNP, we use the Pytorch Geometric library implementations [10]. And we use the implementation released by the authors for other baselines.

- **Pytorch Geometric library:** https://github.com/pyg-team/pytorch_geometric/tree/master/benchmark
- **GPR-GNN:** https://github.com/jianhao2016/GPRGNN
- **BernNet:** https://github.com/ivam-he/BernNet
- **GCNII:** https://github.com/chennnM/GCNII
- **TWIRLS:** https://github.com/FFTYYY/TWIRLS

Table 8: The number of parameters for each model on Cora.

ChebNet(2)	ChebNet(10)	GCN	GPR-GNN	BernNet	ChabBase
138.0k	230.4k	92.1k	92.1k	92.1k	92.1k

Table 9: The hyper-parameters of baselines.

Method	Hyper-parameters
MLP	layers:2, hidden:64, lr:{0.01,0.05}, L_2: {0.0005,0.0}, dropout: 0.5
GCN	layers:2, hidden:64, lr:{0.01,0.05}, L_2: {0.0005,0.0}, dropout: 0.5
APPNP	layers:2, hidden:64, lr:{0.01,0.05}, L_2: {0.0005,0.0}, dropout: 0.5, α: {0.1, 0.2, 0.5, 0.9}, K:10
ARMA	layers:2, hidden:64, lr:{0.01,0.05}, L_2: {0.0005,0.0}, stacks:2, ARMA_layers: 1, skip_dropout: {0.25,0.75}, dropout: 0.5
ChebNet	layers:2, hidden:32, lr:{0.01,0.05}, L_2: {0.0005,0.0}, K:2, dropout: 0.5
GPRGNN	layers:2, hidden:64, K:10, Init:PPR, lr_l:{0.01,0.05}, lr_p: {0.01,0.05}, α: {0.1, 0.2, 0.5, 0.9}, dropout_l: 0.5, dropout_p: {0.0,0.5,0.7}, L_{2_l}: {0.0005,0.0}, L_{2_p}: 0.0
BernNet	layers:2, hidden:64, K:10, L_2: {0.0005,0.0}, lr_l:{0.01,0.05}, lr_p: {0.001,0.002,0.01,0.05}, dropout_l: 0.5, dropout_p: {0.0,0.5,0.6,0.7,0.9},

D.1 Experiments in section 4

Setting. For ChebNet, we set the hidden units as 32 for the propagation steps $K = 2$ and set the hidden units as 16 for the propagation steps $K = 10$ and use 2 convolutional layers. For GCN, we use 2 convolutional layers with 64 hidden units. For GPR-GNN and BernNet, we use 2-layer MLP with 64 hidden units and set the propagation steps $K = 10$. Following the released code, we use PPR to initialize the coefficients w_k's with $\alpha = 0.1$ for GPR-GNN, and initialized $w_k = 1.0$ for BernNet. For ChebBase, we also utilize 2-layer MLP with 64 hidden units and set the propagation steps $K = 10$ like GPR-GNN and BernNet. we set $w_0 = 1.0$, $w_k = 0.0$ for $k \neq 0$ to initialize the Chebyshev coefficients, corresponding to an initial state with no feature propagation. For other hyperparameter tuning, we set the learning rate as 0.01, optimize weight decay over $\{0.0005, 0.05\}$ and dropout over $0.5, 0.8$ for all models.

Our setup ensures that each model's parameters are as exact as possible for fairness in comparison. Table 8 displays the number of parameters for each method on the Cora dataset, with the K for ChebNet indicated by the numbers in parentheses. We can observe that the number of parameters for each model is close to 100k. Note that with $K = 10$, ChebNet's hidden units are already equivalent to 16, and if we keep reducing them, the results will deteriorate.

D.2 Semi-supervised node classification with polynomial based methods

Setup. We list the hyper-parameters for baselines in Table 9, where lr represents the learning rate and L_2 denotes the weight decay. For ARMA, we set the number of parallel stacks as 2, use 1 ARMA layer and optimize the dropout probability of the skip connection over $\{0.25, 0.75\}$, following [2]. For GPR-GNN and BernNet, we use lr_l, L_{2_l} and dropout_l to denote the learning rate, weight decay and dropout for linear layer, respectively, and use lr_p, L_{2_p} and dropout_p for propagation layer. We optimize these parameters according to the reports in their paper [6, 15].

For ChebNetII, we use a 2-layers MLP with 64 hidden units for linear layer as the same as GPR-GNN and BernNet. Table 10 displays the hyper-parameters for ChebNetII for the semi-supervised node classification task, and we optimize the parameters for the linear and propagation layers, respectively.

Results of fixed splits. Table 7 exhibits the results of semi-supervised node classification in fixed splits. We can find that ChebNetII performs essentially the same as the random splits, reaching the best results in seven of the eight datasets, which further demonstrates the expression of the Chebyshev basis.

Table 10: The hyper-parameters of ChebNetII for semi-supervised learning.

Dataset	Hyper-parameters
Cora	layers:2, hidden:64, K:5, lr$_l$:0.01, L_{2_l}:0.0005, dropout$_l$: 0.5, lr$_p$:0.005, L_{2_p}:0.0005, dropout$_p$: 0.5
Citeseer	layers:2, hidden:64, K:10, lr$_l$:0.01, L_{2_l}:0.0005, dropout$_l$: 0.5, lr$_p$:0.001, L_{2_p}:0.05, dropout$_p$: 0.5
Pubmed	layers:2, hidden:64, K:10, lr$_l$:0.01, L_{2_l}:0.0005, dropout$_l$: 0.5, lr$_p$:0.005, L_{2_p}:0.05, dropout$_p$: 0.5
Chameleon	layers:2, hidden:64, K:10, lr$_l$:0.01, L_{2_l}:0.0005, dropout$_l$: 0.5, lr$_p$:0.0005, L_{2_p}:0.0, dropout$_p$: 0.5
Squirrel	layers:2, hidden:64, K:10, lr$_l$:0.01, L_{2_l}:0.0005, dropout$_l$: 0.5, lr$_p$:0.0005, L_{2_p}:0.0, dropout$_p$: 0.5
Actor	layers:2, hidden:64, K:10, lr$_l$:0.01, L_{2_l}:0.0005, dropout$_l$: 0.5, lr$_p$:0.01, L_{2_p}:0.0005, dropout$_p$: 0.9
Texas	layers:2, hidden:64, K:10, lr$_l$:0.01, L_{2_l}:0.0005, dropout$_l$: 0.6, lr$_p$:0.001, L_{2_p}:0.0, dropout$_p$: 0.7
Cornell	layers:2, hidden:64, K:10, lr$_l$:0.01, L_{2_l}:0.0005, dropout$_l$: 0.6, lr$_p$:0.001, L_{2_p}:0.0005, dropout$_p$: 0.7

Table 11: The hyper-parameters of ChebNetII for full-supervised learning

Dataset	Hyper-parameters
Cora	layers:2, hidden:64, K:10, lr$_l$:0.01, L_{2_l}:0.0005, dropout$_l$: 0.5, lr$_p$:0.01, L_{2_p}:0.0005, dropout$_p$: 0.0
Citeseer	layers:2, hidden:64, K:10, lr$_l$:0.01, L_{2_l}:0.0005, dropout$_l$: 0.5, lr$_p$:0.01, L_{2_p}:0.0005, dropout$_p$: 0.0
Pubmed	layers:2, hidden:64, K:10, lr$_l$:0.01, L_{2_l}:0.0, dropout$_l$: 0.5, lr$_p$:0.01, L_{2_p}:0.0, dropout$_p$: 0.0
Chameleon	layers:2, hidden:64, K:10, lr$_l$:0.05, L_{2_l}:0.0, dropout$_l$: 0.5, lr$_p$:0.01, L_{2_p}:0.0, dropout$_p$: 0.6
Squirrel	layers:2, hidden:64, K:10, lr$_l$:0.05, L_{2_l}:0.0, dropout$_l$: 0.5, lr$_p$:0.01, L_{2_p}:0.0, dropout$_p$: 0.5
Actor	layers:2, hidden:64, K:10, lr$_l$:0.05, L_{2_l}:0.0, dropout$_l$: 0.5, lr$_p$:0.01, L_{2_p}:0.0, dropout$_p$: 0.9
Texas	layers:2, hidden:64, K:10, lr$_l$:0.05, L_{2_l}:0.0005, dropout$_l$: 0.6, lr$_p$:0.001, L_{2_p}:0.0, dropout$_p$: 0.5
Cornell	layers:2, hidden:64, K:10, lr$_l$:0.05, L_{2_l}:0.0005, dropout$_l$: 0.5, lr$_p$:0.001, L_{2_p}:0.0005, dropout$_p$: 0.6

D.3 Full-supervised node classification

For the baselines in the prior, we use the same model settings and hyper-parameter optimization as for the semi-supervised node classification task, with the hyper-parameter settings shown in Table 9. For GCNII [4] and TWIRLS [38], we use the setting of hyper-parameters as reported in their papers for the full-supervised task. For ChebnetII, we used the same settings as for the semi-supervised task, and Table 11 exhibits the hyper-parameters for ChebNetII on the full-supervised tasks.