Simple and Deep Graph Convolutional Networks

Ming Chen [1] **Zhewei Wei** [2 3 4] **Zengfeng Huang** [5] **Bolin Ding** [6] **Yaliang Li** [6]

Abstract

Graph convolutional networks (GCNs) are a powerful deep learning approach for graph-structured data. Recently, GCNs and subsequent variants have shown superior performance in various application areas on real-world datasets. Despite their success, most of the current GCN models are shallow, due to the *over-smoothing* problem. In this paper, we study the problem of designing and analyzing deep graph convolutional networks. We propose the GCNII, an extension of the vanilla GCN model with two simple yet effective techniques: *Initial residual* and *Identity mapping*. We provide theoretical and empirical evidence that the two techniques effectively relieves the problem of over-smoothing. Our experiments show that the deep GCNII model outperforms the state-of-the-art methods on various semi- and full-supervised tasks. Code is available at https://github.com/chennnM/GCNII.

1. Introduction

Graph convolutional networks (GCNs) (Kipf & Welling, 2017) generalize convolutional neural networks (CNNs) (LeCun et al., 1995) to graph-structured data. To learn the graph representations, the "graph convolution" operation applies the same linear transformation to all the neighbors of a node followed by a nonlinear activation function. In recent years, GCNs and their variants (Defferrard et al., 2016; Veličković et al., 2018) have been successfully applied to a wide range of applications, including social analysis (Qiu et al., 2018; Li & Goldwasser, 2019), traffic prediction (Guo et al., 2019; Li et al., 2019), biology (Fout et al., 2017; Shang et al., 2019), recommender systems (Ying et al., 2018), and computer vision (Zhao et al., 2019; Ma et al., 2019).

Despite their enormous success, most of the current GCN models are shallow. Most of the recent models, such as GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018), achieve their best performance with 2-layer models. Such shallow architectures limit their ability to extract information from high-order neighbors. However, stacking more layers and adding non-linearity tends to degrade the performance of these models. Such a phenomenon is called *over-smoothing* (Li et al., 2018b), which suggests that as the number of layers increases, the representations of the nodes in GCN are inclined to converge to a certain value and thus become indistinguishable. ResNet (He et al., 2016) solves a similar problem in computer vision with *residual connections*, which is effective for training very deep neural networks. Unfortunately, adding residual connections in the GCN models merely slows down the over-smoothing problem (Kipf & Welling, 2017); deep GCN models are still outperformed by 2-layer models such as GCN or GAT.

Recently, several works try to tackle the problem of over-smoothing. JKNet (Xu et al., 2018) uses dense skip connections to combine the output of each layer to preserve the locality of the node representations. Recently, DropEdge (Rong et al., 2020) suggests that by randomly removing out a few edges from the input graph, one can relieve the impact of over-smoothing. Experiments (Rong et al., 2020) suggest that the two methods can slow down the performance drop as we increase the network depth. However, for semi-supervised tasks, the state-of-the-art results are still achieved by the shallow models, and thus the benefit brought by increasing the network depth remains in doubt.

On the other hand, several methods combine deep propagation with shallow neural networks. SGC (Wu et al., 2019) attempts to capture higher-order information in the graph by applying the K-th power of the graph convolution matrix in a single neural network layer. PPNP and APPNP (Klicpera et al., 2019a) replace the power of the graph convolution matrix with the Personalized PageRank matrix to solve the over-smoothing problem. GDC (Klicpera et al., 2019b) further extends APPNP by generalizing Personalized PageRank (Page et al., 1999) to an arbitrary graph diffusion process. However, these methods perform a linear combination of neighbor features in each layer and lose the

[1]School of Information, Renmin University of China [2]Gaoling School of Artificial Intelligence, Renmin University of China [3]Beijing Key Lab of Big Data Management and Analysis Methods [4]MOE Key Lab of Data Engineering and Knowledge Engineering [5]School of Data Science, Fudan University [6]Alibaba Group. Correspondence to: Zhewei Wei <zhewei@ruc.edu.cn>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

powerful expression ability of deep nonlinear architectures, which means they are still shallow models.

In conclusion, it remains an open problem to design a GCN model that effectively prevents over-smoothing and achieves state-of-the-art results with truly deep network structures. Due to this challenge, it is even unclear whether the network depth is a resource or a burden in designing new graph neural networks. In this paper, we give a positive answer to this open problem by demonstrating that the vanilla GCN (Kipf & Welling, 2017) can be extended to a deep model with two simple yet effective modifications. In particular, we propose **G**raph **C**onvolutional **N**etwork via **I**nitial residual and **I**dentity mapping (**GCNII**), a deep GCN model that resolves the over-smoothing problem. At each layer, initial residual constructs a skip connection from the input layer, while identity mapping adds an identity matrix to the weight matrix. The empirical study demonstrates that the two surprisingly simple techniques prevent over-smoothing and improve the performance of GCNII consistently as we increase its network depth. In particular, the deep GCNII model achieves new state-of-the-art results on various semi-supervised and full-supervised tasks.

Second, we provide theoretical analysis for multi-layer GCN and GCNII models. It is known (Wu et al., 2019) that by stacking k layers, the vanilla GCN essentially simulates a K-th order of polynomial filter with predetermined coefficients. (Wang et al., 2019) points out that such a filter simulates a *lazy random walk* that eventually converges to the stationary vector and thus leads to over-smoothing. On the other hand, we prove that a K-layer GCNII model can express a polynomial spectral filter of order K with arbitrary coefficients. This property is essential for designing deep neural networks. We also derive the closed-form of the stationary vector and analyze the rate of convergence for the vanilla GCN. Our analysis implies that nodes with high degrees are more likely to suffer from over-smoothing in a multi-layer GCN model, and we perform experiments to confirm this theoretical conjecture.

2. Preliminaries

Notations. Given a simple and connected undirected graph $G = (V, E)$ with n nodes and m edges. We define the *self-looped graph* $\tilde{G} = (V, \tilde{E})$ to be the graph with a self-loop attached to each node in G. We use $\{1, \dots, n\}$ to denote the node IDs of G and $\tilde{G}$, and d_j and $d_j + 1$ to denote the degree of node j in G and $\tilde{G}$, respectively. Let $\mathbf{A}$ denote the adjacency matrix and $\mathbf{D}$ the diagonal degree matrix. Consequently, the adjacency matrix and diagonal degree matrix of $\tilde{G}$ is defined to be $\tilde{\mathbf{A}} = \mathbf{A}+\mathbf{I}$ and $\tilde{\mathbf{D}} = \mathbf{D}+\mathbf{I}$, respectively. Let $\mathbf{X} \in \mathbf{R}^{n\times d}$ denote the node feature matrix, that is, each node v is associated with a d-dimensional feature vector $\mathbf{X}_v$. The *normalized graph Laplacian matrix* is defined as $\mathbf{L} = \mathbf{I}_n - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, which is a symmetric positive semidefinite matrix with eigendecomposition $\mathbf{U}\Lambda\mathbf{U}^T$,. Here Λ is a diagonal matrix of the eigenvalues of $\mathbf{L}$, and $\mathbf{U} \in \mathbf{R}^{n\times n}$ is a unitary matrix that consists of the eigenvectors of $\mathbf{L}$. The graph convolution operation between signal $\mathbf{x}$ and filter $g_\gamma(\Lambda) = \mathrm{diag}(\gamma)$ is defined as $g_\gamma(\mathbf{L}) * \mathbf{x} = \mathbf{U}g_\gamma(\Lambda)\mathbf{U}^T\mathbf{x}$, where the parameter $\gamma \in \mathbf{R}^n$ corresponds to a vector of spectral filter coefficients.

Vanilla GCN. (Kipf & Welling, 2017) and (Defferrard et al., 2016) suggest that the graph convolution operation can be further approximated by the K-th order polynomial of Laplacians

$$\mathbf{U}g_\theta(\Lambda)\mathbf{U}^T\mathbf{x} \approx \mathbf{U}\left(\sum_{\ell=0}^{K}\theta_\ell\mathbf{\Lambda}^\ell\right)\mathbf{U}^\top\mathbf{x} = \left(\sum_{\ell=0}^{K}\theta_\ell\mathbf{L}^\ell\right)\mathbf{x},$$

where $\theta \in \mathbf{R}^{K+1}$ corresponds to a vector of polynomial coefficients. The vanilla GCN (Kipf & Welling, 2017) sets $K = 1$, $\theta_0 = 2\theta$ and $\theta_1 = -\theta$ to obtain the convolution operation $\mathbf{g}_\theta * \mathbf{x} = \theta\left(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)\mathbf{x}$. Finally, by *the renormalization trick*, (Kipf & Welling, 2017) replaces the matrix $\mathbf{I}+\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ by a normalized version $\tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2} = (\mathbf{D}+\mathbf{I}_n)^{-1/2}(\mathbf{A}+\mathbf{I}_n)(\mathbf{D}+\mathbf{I}_n)^{-1/2}$. and obtains the Graph Convolutional Layer

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

Where σ denotes the ReLU operation.

SGC (Wu et al., 2019) shows that by stacking K layers, GCN corresponds to a *fixed polynomial filter* of order K on the graph spectral domain of $\tilde{G}$. In particular, let $\tilde{\mathbf{L}} = \mathbf{I}_n - \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ denote the normalized graph Laplacian matrix of the self-looped graph $\tilde{G}$. Consequently, applying a K-layer GCN to a signal $\mathbf{x}$ corresponds to $\left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\right)^K\mathbf{x} = \left(\mathbf{I}_n - \tilde{\mathbf{L}}\right)^K\mathbf{x}$. (Wu et al., 2019) also shows that by adding a self-loop to each node, $\tilde{\mathbf{L}}$ effectively shrinks the underlying graph spectrum.

APPNP. (Klicpera et al., 2019a) uses Personalized PageRank to derive a fixed filter of order K. Let $f_\theta(\mathbf{X})$ denote the output of a two-layer fully connected neural network on the feature matrix $\mathbf{X}$, PPNP's model is defined as

$$\mathbf{H} = \alpha\left(\mathbf{I}_n - (1-\alpha)\tilde{\mathbf{A}}\right)^{-1}f_\theta(\mathbf{X}). \tag{2}$$

Due to the property of Personalized PageRank, such a filter preserves locality and thus is suitable for classification tasks. (Klicpera et al., 2019a) also proposes APPNP, which replaces $\alpha\left(\mathbf{I}_n - (1-\alpha)\tilde{\mathbf{A}}\right)^{-1}$ with an approximation derived by a truncated power iteration. Formally, APPNP with K-hop aggregation is defined as

$$\boldsymbol{H}^{(\ell+1)} = (1-\alpha)\tilde{\boldsymbol{P}}\boldsymbol{H}^{(\ell)} + \alpha\boldsymbol{H}^{(0)}, \tag{3}$$

where $\boldsymbol{H}^{(0)} = f_\theta(\boldsymbol{X})$. By decoupling feature transformation and propagation, PPNP and APPNP can aggregate information from multi-hop neighbors without increasing the number of layers in the neural network.

JKNet. The first deep GCN framework is proposed by (Xu et al., 2018). At the last layer, JKNet combines all previous representations $\left[\mathbf{H}^{(1)}, \ldots, \mathbf{H}^{(K)}\right]$ to learn representations of different orders for different graph substructures. (Xu et al., 2018) proves that 1) a K-layer vanilla GCN model simulates random walks of K steps in the self-looped graph $\tilde{G}$ and 2) by combining all representations from the previous layers, JKNet relieves the problem of over-smoothing.

DropEdge A recent work (Rong et al., 2020) suggests that randomly removing some edges from $\tilde{G}$ retards the convergence speed of over-smoothing. Let $\tilde{\mathbf{P}}_{\text{drop}}$ denote the renormalized graph convolution matrix with some edge removed at random, the vanilla GCN equipped with DropEdge is defined as

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

3. GCNII Model

It is known (Wu et al., 2019) that by stacking K layers, the vanilla GCN simulates a polynomial filter $\left(\sum_{\ell=0}^{K}\theta_\ell\tilde{\mathbf{L}}^\ell\right)\mathbf{x}$ of order K with fixed coefficients θ on the graph spectral domain of $\tilde{G}$. The fixed coefficients limit the expressive power of a multi-layer GCN model and thus leads to over-smoothing. To extend GCN to a truly deep model, we need to enable GCN to express a K order polynomial filter with *arbitrary* coefficients. We show this can be achieved by two simple techniques: *Initial residual connection* and *Identity mapping*. Formally, we define the ℓ-th layer of GCNII as

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\left((1-\alpha_\ell)\tilde{\mathbf{P}}\mathbf{H}^{(\ell)} + \alpha_\ell\mathbf{H}^{(0)}\right)\left((1-\beta_\ell)\mathbf{I}_n + \beta_\ell\mathbf{W}^{(\ell)}\right)\right), \tag{5}$$

where α_ℓ and β_ℓ are two hyperparameters to be discussed later. Recall that $\tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ is the graph convolution matrix with the renormalization trick. Note that compared to the vanilla GCN model (equation (1)), we make two modifications: 1) We combine the smoothed representation $\tilde{\mathbf{P}}\mathbf{H}^{(\ell)}$ with an initial residual connection to the first layer $\mathbf{H}^{(0)}$; 2) We add an identity mapping $\mathbf{I}_n$ to the ℓ-th weight matrix $\mathbf{W}^{(\ell)}$.

Initial residual connection. To simulate the skip connection in ResNet (He et al., 2016), (Kipf & Welling, 2017) proposes *residual connection* that combines the smoothed representation $\tilde{\mathbf{P}}\mathbf{H}^{(\ell)}$ with $\mathbf{H}^{(\ell)}$. However, it is also shown in (Kipf & Welling, 2017) that such residual connection only partially relieves the over-smoothing problem; the performance of the model still degrades as we stack more layers.

We propose that, instead of using a residual connection to carry the information from the previous layer, we construct a connection to the initial representation $\mathbf{H}^{(0)}$. The initial residual connection ensures that that the final representation of each node retains at least a fraction of α_ℓ from the input layer even if we stack many layers. In practice, we can simply set $\alpha_\ell = 0.1$ or 0.2 so that the final representation of each node consists of at least a fraction of the input feature. We also note that $\mathbf{H}^{(0)}$ does not necessarily have to be the feature matrix $\mathbf{X}$. If the feature dimension d is large, we can apply a fully-connected neural network on $\mathbf{X}$ to obtain a lower-dimensional initial representation $\mathbf{H}^{(0)}$ before the forward propagation.

Finally, we recall that APPNP (Klicpera et al., 2019a) employs a similar approach to the initial residual connection in the context of Personalized PageRank. However, (Klicpera et al., 2019a) also shows that performing multiple nonlinearity operations to the feature matrix will lead to overfitting and thus results in the performance drop. Therefore, APPNP applies a linear combination between different layers and thus remains a shallow model. This suggests that the idea of initial residual alone is not sufficient to extend GCN to a deep model.

Identity mapping. To amend the deficiency of APPNP, we borrow the idea of identity mapping from ResNet. At the ℓ-th layer, we add an identity matrix $\mathbf{I}_n$ to the weight matrix $\mathbf{W}^{(\ell)}$. In the following, we summarize the motivations for introducing identity mapping into our model.

- Similar to the motivation of ResNet (He et al., 2016), identity mapping ensures that a deep GCNII model achieves at least the same performance as its shallow version does. In particular, by setting β_ℓ sufficiently small, deep GCNII ignores the weight matrix $\mathbf{W}^{(\ell)}$ and essentially simulates APPNP (equation (3)).
- It has been observed that frequent interaction between different dimensions of the feature matrix (Klicpera et al., 2019a) degrades the performance of the model in semi-supervised tasks. Mapping the smoothed representation $\tilde{\mathbf{P}}\mathbf{H}^{(\ell)}$ directly to the output reduces such interaction.
- Identity mapping is proved to be particularly useful in semi-supervised tasks. It is shown in (Hardt & Ma, 2017) that a linear ResNet of the form $\mathbf{H}^{(\ell+1)} = \mathbf{H}^{(\ell)}\left(\mathbf{W}^{(\ell)} + \mathbf{I}_n\right)$ satisfies the following properties: 1) The optimal weight matrices $\mathbf{W}^{(l)}$ have small norms; 2) The only critical point is the global minimum. The first property allows us to put strong regularization on

$\mathbf{W}^{\ell}$ to avoid over-fitting, while the later is desirable in semi-supervised tasks where training data is limited.

- (Oono & Suzuki, 2020) theoretically proves that the node features of a K-layer GCNs will converge to a subspace and incur information loss. In particular, the rate of convergence depends on s^K, where s is the maximum singular value of the weight matrices $\mathbf{W}^{(\ell)}, \ell = 0, \ldots, K-1$. By replacing $\mathbf{W}^{(\ell)}$ with $(1-\beta_\ell)\mathbf{I}_n + \beta_\ell \mathbf{W}^{(\ell)}$ and imposing regularization on $\mathbf{W}^{(\ell)}$, we force the norm of $\mathbf{W}^{(\ell)}$ to be small. Consequently, the singular values of $(1-\beta_\ell)\mathbf{I}_n + \beta_\ell \mathbf{W}^{(\ell)}$ will be close to 1. Therefore, the maximum singular value s will also be close to 1, which implies that s^K is large, and the information loss is relieved.

The principle of setting β_ℓ is to ensure the decay of the weight matrix adaptively increases as we stack more layers. In practice, we set $\beta_\ell = \log(\frac{\lambda}{\ell} + 1) \approx \frac{\lambda}{\ell}$, where λ is a hyperparameter.

Connection to iterative shrinkage-thresholding. Recently, there has been work on optimization-inspired network structure design (Zhang & Ghanem, 2018; Papyan et al., 2017). The idea is that a feedforward neural network can be considered as an iterative optimization algorithm to minimize some function, and it was hypothesized that better optimization algorithms might lead to better network structure (Li et al., 2018a). Thus, theories in numerical optimization algorithms may inspire the design of better and more interpretable network structures. As we will show next, the use of identity mappings in our structure is also well-motivated from this. We consider the LASSO objective:

$$\min_{x \in \mathcal{R}^n} \frac{1}{2}\|\mathbf{B}\mathbf{x} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{x}\|_1.$$

Similar to compressive sensing, we consider $\mathbf{x}$ as the signal we are trying to recover, $\mathbf{B}$ as the measurement matrix, and $\mathbf{y}$ as the signal we observe. In our setting, $\mathbf{y}$ is the original feature of a node, and $\mathbf{x}$ is the node embedding the network tries to learn. As opposed to standard regression models, the design matrix $\mathbf{B}$ is unknown parameters and will be learned through back propagation. So, this is in the same spirit as the sparse coding problem, which has been used to design and to analyze CNNs (Papyan et al., 2017). Iterative shrinkage-thresholding algorithms are effective for solving the above optimization problem, in which the update in the $(t+1)$th iteration is:

$$\mathbf{x}^{t+1} = P_{\mu_t \lambda}\left(\mathbf{x}^t - \mu_t \mathbf{B}^T \mathbf{B}\mathbf{x}^t + \mu_t \mathbf{B}^T \mathbf{y}\right),$$

Here μ_t is the step size, and $P_\beta(\cdot)$ (with $\beta > 0$) is the entry-wise soft thresholding function:

$$P_\theta(z) = \begin{cases} z - \theta, & \text{if } z \geq \theta \\ 0, & \text{if } |z| < \theta \\ z + \theta, & \text{if } z \leq -\theta \end{cases}.$$

Now, if we reparameterize $-\mathbf{B}^\mathbf{T}\mathbf{B}$ by $\mathbf{W}$, the above update formula becomes quite similar to the one used in our method. More spopposeecifically, we have $\mathbf{x}^{t+1} = P_{\mu_t \lambda}\left((\mathbf{I} + \mu_t \mathbf{W})\mathbf{x}^t + \mu_t \mathbf{B}^T \mathbf{y}\right)$, where the term $\mu_t \mathbf{B}^T \mathbf{y}$ corresponds to the initial residual, and $\mathbf{I} + \mu_t \mathbf{W}$ corresponds to the identity mapping in our model (5). The soft thresholding operator acts as the nonlinear activation function, which is similar to the effect of ReLU activation. In conclusion, our network structure, especially the use of identity mapping is well-motivated from iterative shrinkage-thresholding algorithms for solving LASSO.

4. Spectral Analysis

4.1. Spectral analysis of multi-layer GCN.

We consider the following GCN model with residual connection:

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\left(\tilde{\mathbf{P}}\mathbf{H}^{(\ell)} + \mathbf{H}^{(\ell)}\right)\mathbf{W}^{(\ell)}\right). \tag{6}$$

Recall that $\tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ is the graph convolution matrix with the renormalization trick. (Wang et al., 2019) points out that equation (6) simulates a *lazy random walk* with the transition matrix $\frac{\mathbf{I}_n + \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}}{2}$. Such a lazy random walk eventually converges to the stationary state and thus leads to over-smoothing. We now derive the closed-form of the stationary vector and analyze the rate of such convergence. Our analysis suggests that the converge rate of an individual node depends on its degree, and we conduct experiments to back up this theoretical finding. In particular, we have the following Theorem.

Theorem 1. *Assume the self-looped graph $\tilde{G}$ is connected. Let $\mathbf{h}^{(K)} = \left(\frac{\mathbf{I}_n + \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}}{2}\right)^K \cdot \mathbf{x}$ denote the representation by applying a K-layer renormalized graph convolution with residual connection to a graph signal $\mathbf{x}$. Let $\lambda_{\tilde{G}}$ denote the spectral gap of the self-looped graph $\tilde{G}$, that is, the least nonzero eigenvalue of the normalized Laplacian $\tilde{\mathbf{L}} = \mathbf{I}_n - \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$. We have*

1) As K goes to infinity, $\mathbf{h}^{(K)}$ converges to $\boldsymbol{\pi} = \frac{\langle \tilde{\mathbf{D}}^{1/2}\mathbf{1}, \mathbf{x}\rangle}{2m+n} \cdot \tilde{\mathbf{D}}^{1/2}\mathbf{1}$, where $\mathbf{1}$ denotes an all-one vector.

2) The convergence rate is determined by

$$\mathbf{h}^{(K)} = \boldsymbol{\pi} \pm \left(\sum_{i=1}^n x_i\right) \cdot \left(1 - \frac{\lambda_{\tilde{G}}^2}{2}\right)^K \cdot \mathbf{1}. \tag{7}$$

Recall that m and n are the number of nodes and edges in the original graph G. We use the operator $\pm$ to indicate that for each entry $\mathbf{h}^{(K)}(j)$ and $\boldsymbol{\pi}(j)$, $j = 1, \ldots, n$,

$$\left|\mathbf{h}^{(K)}(j) - \boldsymbol{\pi}(j)\right| \leq \left(\sum_{i=1}^n x_i\right) \cdot \left(1 - \frac{\lambda_{\tilde{G}}^2}{2}\right)^K.$$

The proof of Theorem 1 can be found in the supplementary materials. There are two consequences from Theorem 1. First of all, it suggests that the K-th representation of GCN $\mathbf{h}^{(K)}$ converges to a vector $\boldsymbol{\pi} = \frac{\langle \tilde{\mathbf{D}}^{1/2}\mathbf{1}, \mathbf{x} \rangle}{2m+n} \cdot \tilde{\mathbf{D}}^{1/2}\mathbf{1}$. Such convergence leads to over-smoothing as the vector $\boldsymbol{\pi}$ only carries the two kinds of information: the degree of each node, and the inner product between the initial signal $\mathbf{x}$ and vector $\mathbf{D}^{1/2}\mathbf{1}$.

Convergence rate and node degree. Equation (7) suggests that the converge rate depends on the summation of feature entries $\sum_{i=1}^{n} x_i$ and the spectral gap $\lambda_{\tilde{G}}$. If we take a closer look at the *relative converge rate* for an individual node j, we can express its final representation $\mathbf{h}^{(K)}(j)$ as

$$\mathbf{h}^{(K)}(j) = \sqrt{d_j + 1}\left(\sum_{i=1}^{n} \frac{\sqrt{d_i+1}}{2m+n} x_i \pm \frac{\sum_{i=1}^{n} x_i \left(1 - \frac{\lambda_{\tilde{G}}^2}{2}\right)^K}{\sqrt{d_j+1}}\right).$$

This suggests that if a node j has a higher degree of d_j (and hence a larger $\sqrt{d_j+1}$), its representation $\mathbf{h}^{(K)}(j)$ converges faster to the stationary state $\boldsymbol{\pi}(j)$. Based on this fact, we make the following conjecture.

Conjecture 1. *Nodes with higher degrees are more likely to suffer from over-smoothing.*

We will verify Conjecture 1 on real-world datasets in our experiments.

4.2. Spectral analysis of GCNII

We consider the spectral domain of the self-looped graph $\tilde{G}$. Recall that a polynomial filter of order K on a graph signal $\mathbf{x}$ is defined as $\left(\sum_{\ell=0}^{K} \theta_\ell \tilde{\mathbf{L}}^\ell\right)\mathbf{x}$, where $\tilde{\mathbf{L}}$ is the normalized Laplacian matrix of $\tilde{G}$ and θ_k's are the polynomial coefficients. (Wu et al., 2019) proves that a K-layer GCN simulates a polynomial filter of order K with fixed coefficients θ. As we shall prove later, such fixed coefficients limit the expressive power of GCN and thus leads to over-smoothing. On the other hand, we show a K-layer GCNII model can express a K order polynomial filter with arbitrary coefficients.

Theorem 2. *Consider the self-looped graph $\tilde{G}$ and a graph signal $\mathbf{x}$. A K-layer GCNII can express a K order polynomial filter $\left(\sum_{\ell=0}^{K} \theta_\ell \tilde{\mathbf{L}}^\ell\right)\mathbf{x}$ with arbitrary coefficients θ.*

The proof of Theorem 2 can be found in the supplementary materials. Intuitively, the parameter β allows GCNII to simulate the coefficient θ_ℓ of the polynomial filter.

Expressive power and over-smoothing. The ability to express a polynomial filter with arbitrary coefficients is essential for preventing over-smoothing. To see why this is the case, recall that Theorem 1 suggests a K-layer vanilla GCN simulates a fixed K-order polynomial filter $\tilde{\mathbf{P}}^K\mathbf{x}$, where $\tilde{\mathbf{P}}$ is the renormalized graph convolution matrix. Over-smoothing is caused by the fact that $\tilde{\mathbf{P}}^K\mathbf{x}$ converges to a distribution isolated from the input feature $\mathbf{x}$ and thus incuring gradient vanishment. DropEdge (Rong et al., 2020) slows down the rate of convergence, but eventually will fail as K goes to infinity.

On the other hand, Theorem 2 suggests that deep GCNII converges to a distribution that carries information from both the input feature and the graph structure. This property alone ensures that GCNII will not suffer from over-smoothing even if the number of layers goes to infinity. More precisely, Theorem 2 states that a K-layer GCNII can express $\mathbf{h}^{(K)} = \left(\sum_{\ell=0}^{K} \theta_\ell \tilde{\mathbf{L}}^\ell\right) \cdot \mathbf{x}$ with arbitrary coefficients θ. Since the renormalized graph convolution matrix $\tilde{\mathbf{P}} = \mathbf{I}_n - \tilde{\mathbf{L}}$, it follows that K-layer GCNII can express $\mathbf{h}^{(K)} = \left(\sum_{\ell=0}^{K} \theta'_\ell \tilde{\mathbf{P}}^\ell\right) \cdot \mathbf{x}$ with arbitrary coefficients θ'. Note that with a proper choice of θ', $\mathbf{h}^{(K)}$ can carry information from both the input feature and the graph structure even with K going to infinity. For example, APPNP (Klicpera et al., 2019a) and GDC (Klicpera et al., 2019b) set $\theta'_i = \alpha(1-\alpha)^i$ for some constant $0 < \alpha < 1$. As K goes to infinity, $\mathbf{h}^{(K)} = \left(\sum_{\ell=0}^{K} \theta'_\ell \tilde{\mathbf{P}}^\ell\right) \cdot \mathbf{x}$ converges to the Personalized PageRank vector of $\mathbf{x}$, which is a function of both the adjacency matrix $\tilde{\mathbf{A}}$ and the input feature vector $\mathbf{x}$. The difference between GCNII and APPNP/GDC is that 1) the coefficient vector theta in our model is learned from the input feature and the label, and 2) we impose a ReLU operation at each layer.

5. Other Related Work

Spectral-based GCN has been extensively studied for the past few years. (Li et al., 2018c) improves flexibility by learning a task-driven adaptive graph for each graph data while training. (Xu et al., 2019) uses the graph wavelet basis instead of the Fourier basis to improve sparseness and locality. Another line of works focuses on the attention-based GCN model (Veličković et al., 2018; Thekumparampil et al., 2018; Zhang et al., 2018), which learn the edge weights at each layer based on node features. (Abu-El-Haija et al., 2019) learn neighborhood mixing relationships by mixing of neighborhood information at various distances but still uses a two-layer model. (Gao & Ji, 2019; Lee et al., 2019) devote to extend pooling operations to graph neural network. For unsupervised information, (Velickovic et al., 2019) train graph convolutional encoder through maximizing mutual information. (Pei et al., 2020) build structural neighborhoods in the latent space of graph embedding for aggregation to extract more structural information. (Dave et al., 2019) uses a single representation vector to capture both topological

Table 1. Dataset statistics.

Dataset	Classes	Nodes	Edges	Features
Cora	7	2,708	5,429	1,433
Citeseer	6	3,327	4,732	3,703
Pubmed	3	19,717	44,338	500
Chameleon	4	2,277	36,101	2,325
Cornell	5	183	295	1,703
Texas	5	183	309	1,703
Wisconsin	5	251	499	1,703
PPI	121	56,944	818,716	50

information and nodal attributes in graph embedding. Many of the sampling-based methods proposed to improve the scalability of GCN. (Hamilton et al., 2017) uses a fixed size of neighborhood samples through layers, (Chen et al., 2018a; Huang et al., 2018) propose efficient variants based on importance sampling. (Chiang et al., 2019) construct minibatch based on graph clustering.

6. Experiments

In this section, we evaluate the performance of GCNII against the state-of-the-art graph neural network models on a wide variety of open graph datasets.

Dataset and experimental setup. We use three standard citation network datasets Cora, Citeseer, and Pubmed (Sen et al., 2008) for semi-supervised node classification. In these citation datasets, nodes correspond to documents, and edges correspond to citations; each node feature corresponds to the bag-of-words representation of the document and belongs to one of the academic topics. For full-supervised node classification, we also include Chameleon (Rozemberczki et al., 2019), Cornell, Texas, and Wisconsin (Pei et al., 2020). These datasets are web networks, where nodes and edges represent web pages and hyperlinks, respectively. The feature of each node is the bag-of-words representation of the corresponding page. For inductive learning, we use Protein-Protein Interaction (PPI) networks (Hamilton et al., 2017), which contains 24 graphs. Following the setting of previous work (Veličković et al., 2018), we use 20 graphs for training, 2 graphs for validation, and the rest for testing. Statistics of the datasets are summarized in Table 1.

Besides GCNII (5), we also include GCNII*, a variant of GCNII that employs different weight matrices for the smoothed representation $\tilde{\mathbf{P}}\mathbf{H}^{(\ell)}$ and the initial residual $\mathbf{H}^{(0)}$. Formally, the $(\ell+1)$-th layer of GCNII* is defined as

$$\mathbf{H}^{(\ell+1)} = \sigma\left((1-\alpha_\ell)\tilde{\mathbf{P}}\mathbf{H}^{(\ell)}\left((1-\beta_\ell)\mathbf{I}_n + \beta_\ell \mathbf{W}_1^{(\ell)}\right) + \right.$$
$$\left. +\alpha_\ell \mathbf{H}^{(0)}\left((1-\beta_\ell)\mathbf{I}_n + \beta_\ell \mathbf{W}_2^{(\ell)}\right)\right).$$

Table 2. Summary of classification accuracy (%) results on Cora, Citeseer, and Pubmed. The number in parentheses corresponds to the number of layers of the model.

Method	Cora	Citeseer	Pubmed
GCN	81.5	71.1	79.0
GAT	83.1	70.8	78.5
APPNP	83.3	71.8	80.1
JKNet	81.1 (4)	69.8 (16)	78.1 (32)
JKNet(Drop)	83.3 (4)	72.6 (16)	79.2 (32)
Incep(Drop)	83.5 (64)	72.7 (4)	79.5 (4)
GCNII	**85.5** $\pm$ **0.5** (64)	**73.4** $\pm$ **0.6** (32)	80.2 $\pm$ 0.4 (16)
GCNII*	85.3 $\pm$ 0.2 (64)	73.2 $\pm$ 0.8 (32)	**80.3** $\pm$ **0.4** (16)

As mentioned in Section 3, we set $\beta_\ell = \log(\frac{\lambda}{\ell}+1) \approx \lambda/\ell$, where λ is a hyperparameter.

6.1. Semi-supervised Node Classification

Setting and baselines. For the semi-supervised node classification task, we apply the standard fixed training/validation/testing split (Yang et al., 2016) on three datasets Cora, Citeseer, and Pubmed, with 20 nodes per class for training, 500 nodes for validation and 1,000 nodes for testing. For baselines, we include two recent deep GNN models: JKNet (Xu et al., 2018) and DropEdge (Rong et al., 2020). As suggested in (Rong et al., 2020), we equip DropEdge on three backbones: GCN (Kipf & Welling, 2017), JKNet (Xu et al., 2018) and IncepGCN (Rong et al., 2020). We also include three state-of-the-art shallow models: GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018) and APPNP (Klicpera et al., 2019a).

We use the Adam SGD optimizer (Kingma & Ba, 2015) with a learning rate of 0.01 and early stopping with a patience of 100 epochs to train GCNII and GCNII*. We set $\alpha_\ell = 0.1$ and L_2 regularization to 0.0005 for the dense layer on all datasets. We perform a grid search to tune the other hyper-parameters for models with different depths based on the accuracy on the validation set. More details of hyper-parameters are listed in the supplementary materials.

Comparison with SOTA. Table 2 reports the mean classification accuracy with the standard deviation on the test nodes of GCN and GCNII after 100 runs. We reuse the metrics already reported in (Fey & Lenssen, 2019) for GCN, GAT, and APPNP, and the best metrics reported in (Rong et al., 2020) for JKNet, JKNet(Drop) and Incep(Drop). Our results successfully demonstrate that GCNII and GCNII* achieves new state-of-the-art performance across all three datasets. Notably, GCNII outperforms the previous state-of-the-art methods by at least 2%. It is also worthwhile to note that the two recent deep models, JKNet and IncepGCN with DropEdge, do not seem to offer significant advantages over the shallow model APPNP. On the other hand, our

Table 3. Summary of classification accuracy (%) results with various depths.

Dataset	Method	Layers 2	4	8	16	32	64
Cora	GCN	**81.1**	80.4	69.5	64.9	60.3	28.7
	GCN(Drop)	**82.8**	82.0	75.8	75.7	62.5	49.5
	JKNet	-	80.2	80.7	80.2	**81.1**	71.5
	JKNet(Drop)	-	**83.3**	82.6	83.0	82.5	83.2
	Incep	-	77.6	76.5	81.7	**81.7**	80.0
	Incep(Drop)	-	82.9	82.5	83.1	83.1	**83.5**
	GCNII	82.2	82.6	84.2	84.6	85.4	**85.5**
	GCNII*	80.2	82.3	82.8	83.5	84.9	**85.3**
Citeseer	GCN	**70.8**	67.6	30.2	18.3	25.0	20.0
	GCN(Drop)	**72.3**	70.6	61.4	57.2	41.6	34.4
	JKNet	-	68.7	67.7	**69.8**	68.2	63.4
	JKNet(Drop)	-	72.6	71.8	**72.6**	70.8	72.2
	Incep	-	69.3	68.4	**70.2**	68.0	67.5
	Incep(Drop)	-	**72.7**	71.4	72.5	72.6	71.0
	GCNII	68.2	68.9	70.6	72.9	**73.4**	73.4
	GCNII*	66.1	67.9	70.6	72.0	**73.2**	73.1
Pubmed	GCN	**79.0**	76.5	61.2	40.9	22.4	35.3
	GCN(Drop)	**79.6**	79.4	78.1	78.5	77.0	61.5
	JKNet	-	78.0	**78.1**	72.6	72.4	74.5
	JKNet(Drop)	-	78.7	78.7	79.1	**79.2**	78.9
	Incep	-	77.7	**77.9**	74.9	OOM	OOM
	Incep(Drop)	-	**79.5**	78.6	79.0	OOM	OOM
	GCNII	78.2	78.8	79.3	**80.2**	79.8	79.7
	GCNII*	77.7	78.2	78.8	**80.3**	79.8	80.1

method achieves this result with a 64-layer model, which demonstrates the benefit of deep network structures.

A detailed comparison with other deep models. Table 3 summaries the results for the deep models with various numbers of layers. We reuse the best-reported results for JKNet, JKNet(Drop) and Incep(Drop) [1]. We observe that on Cora and Citeseer, the performance of GCNII and GCNII* consistently improves as we increase the number of layers. On Pubmed, GCNII and GCNII* achieve the best results with 16 layers, and maintain similar performance as we increase the network depth to 64. We attribute this quality to the identity mapping technique. Overall, the results suggest that with initial residual and identity mapping, we can resolve the over-smoothing problem and extend the vanilla GCN into a truly deep model. On the other hand, the performance of GCN with DropEdge and JKNet drops rapidly as the number of layers exceeds 32, which means they still suffer from over-smoothing.

6.2. Full-Supervised Node Classification

We now evaluate GCNII in the task of full-supervised node classification. Following the setting in (Pei et al., 2020), we use 7 datasets: Cora, Citeseer, Pubmed, Chameleon, Cornell, Texas, and Wisconsin. For each datasets, we randomly split nodes of each class into 60%, 20%, and 20% for training, validation and testing, and measure the performance of all models on the test sets over 10 random splits, as suggested in (Pei et al., 2020). We fix the learning rate to 0.01, dropout rate to 0.5 and the number of hidden units to 64 on all datasets and perform a hyper-parameter search to tune other hyper-parameters based on the validation set. Detailed configuration of all model for full-supervised node classification can be found in the supplementary materials. Besides the previously mentioned baselines, we also include three variants of Geom-GCN (Pei et al., 2020) as they are the state-of-the-art models on these datasets.

[1] https://github.com/DropEdge/DropEdge

Table 4. Summary of Micro-averaged F1 scores on PPI.

Method	PPI
GraphSAGE (Hamilton et al., 2017)	61.2
VR-GCN (Chen et al., 2018b)	97.8
GaAN (Zhang et al., 2018)	98.71
GAT (Veličković et al., 2018)	97.3
JKNet (Xu et al., 2018)	97.6
GeniePath (Liu et al., 2019)	98.5
Cluster-GCN (Chiang et al., 2019)	99.36
GCNII	99.53 ± 0.01
GCNII*	$\mathbf{99.56 \pm 0.02}$

Table 5 reports the mean classification accuracy of each model. We reuse the metrics already reported in (Pei et al., 2020) for GCN, GAT, and Geom-GCN. We observe that GCNII and GCNII* achieves new state-of-the-art results on 6 out of 7 datasets, which demonstrates the superiority of the deep GCNII framework. Notably, GCNII* outperforms APPNP by over 12% on the Wisconsin dataset. This result suggests that by introducing non-linearity into each layer, the predictive power of GCNII is stronger than that of the linear model APPNP.

6.3. Inductive Learning

For the inductive learning task, we apply 9-layer GCNII and GCNII* models with 2048 hidden units on the PPI dataset. We fix the following sets of hyperparameters: $\alpha_\ell = 0.5$, $\lambda = 1.0$ and learning rate of 0.001. Due to the large volume of training data, we set the dropout rate to 0.2 and the weight decay to zero. Following (Veličković et al., 2018), we also add a skip connection from the ℓ-th layer to the $(\ell+1)$-th layer of GCNII and GCNII* to speed up the convergence of the training process. We compare GCNII with the following state-of-the-art methods: GraphSAGE (Hamilton et al., 2017), VR-GCN (Chen et al., 2018b), GaAN (Zhang et al., 2018), GAT (Veličković et al., 2018), JKNet (Xu et al.,

Table 5. Mean classification accuracy of full-supervised node classification.

Method	Cora	Cite.	Pumb.	Cham.	Corn.	Texa.	Wisc.
GCN	85.77	73.68	88.13	28.18	52.70	52.16	45.88
GAT	86.37	74.32	87.62	42.93	54.32	58.38	49.41
Geom-GCN-I	85.19	**77.99**	90.05	60.31	56.76	57.58	58.24
Geom-GCN-P	84.93	75.14	88.09	60.90	60.81	67.57	64.12
Geom-GCN-S	85.27	74.71	84.75	59.96	55.68	59.73	56.67
APPNP	87.87	76.53	89.40	54.3	73.51	65.41	69.02
JKNet	85.25 (16)	75.85 (8)	88.94 (64)	60.07 (32)	57.30 (4)	56.49 (32)	48.82 (8)
JKNet(Drop)	87.46 (16)	75.96 (8)	89.45 (64)	62.08 (32)	61.08 (4)	57.30 (32)	50.59 (8)
Incep(Drop)	86.86 (8)	76.83 (8)	89.18 (4)	61.71 (8)	61.62 (16)	57.84 (8)	50.20 (8)
GCNII	**88.49** (64)	77.08 (64)	89.57 (64)	60.61 (8)	74.86 (16)	69.46 (32)	74.12 (16)
GCNII*	88.01 (64)	77.13 (64)	**90.30** (64)	**62.48** (8)	**76.49** (16)	**77.84** (32)	**81.57** (16)

2018), GeniePath (Liu et al., 2019), Cluster-GCN (Chiang et al., 2019). The metrics are summarized in Table 4.

In concordance with our expectations, the results show that GCNII and GCNII* achieve new state-of-the-art performance on PPI. In particular, GCNII achieves this performance with a 9-layer model, while the number of layers with all baseline models are less or equal to 5. This suggests that larger predictive power can also be leveraged by increasing the network depth in the task of inductive learning.

6.4. Over-Smoothing Analysis for GCN

Recall that Conjecture 1 suggests that nodes with higher degrees are more likely to suffer from over-smoothing. To verify this conjecture, we study how the classification accuracy varies with node degree in the semi-supervised node classification task on Cora, Citeseer, and Pubmed. More specifically, we group the nodes of each graph according to their degrees. The i-th group consists of nodes with degrees in the range $[2^i, 2^{i+1})$ for $i = 0, \ldots, \infty$. For each group, we report the average classification accuracy of GCN with residual connection with various network depths in Figure 1.

We have the following observations. First of all, we note that the accuracy of the 2-layer GCN model increases with the node degree. This is as expected, as nodes with higher degrees generally gain more information from their neighbors. However, as we extend the network depth, the accuracy of high-degree nodes drops more rapidly than that of low-degree nodes. Notably, GCN with 64 layers is unable to classify nodes with degrees larger than 100. This suggests that over-smoothing indeed has a greater impact on nodes with higher degrees.

6.5. Ablation Study

Figure 2 shows the results of an ablation study that evaluates the contributions of our two techniques: initial residual connection and identity mapping. We make three observations from Figure 2: 1) Directly applying identity mapping to the vanilla GCN retards the effect of over-smoothing marginally. 2) Directly applying initial residual connection to the vanilla GCN relieves over-smoothing significantly. However, the best performance is still achieved by the 2-layer model. 3) Applying identity mapping and initial residual connection simultaneously ensures that the accuracy increases with the network depths. This result suggests that both techniques are needed to solve the problem of over-smoothing.

7. Conclusion

We propose GCNII, a simple and deep GCN model that prevents over-smoothing by initial residual connection and identity mapping. The theoretical analysis shows that GCNII is able to express a K order polynomial filter with arbitrary coefficients. For vanilla GCN with multiple layers, we provide theoretical and empirical evidence that nodes with higher degrees are more likely to suffer from over-smoothing. Experiments show that the deep GCNII model achieves new state-of-the-art results on various semi- and full-supervised tasks. Interesting directions for future work include combining GCNII with the attention mechanism and analyzing the behavior of GCNII with the ReLU operation.

Acknowledgements

This research was supported in part by National Natural Science Foundation of China (No. 61832017, No. 61932001 and No. 61972401), by Beijing Outstanding Young Scientist Program NO. BJJWZYJH012019100020098, by the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 18XNLG21, by Shanghai Science and Technology Commission (Grant No. 17JC1420200), by Shanghai Sailing Program (Grant No. 18YF1401200) and a research fund

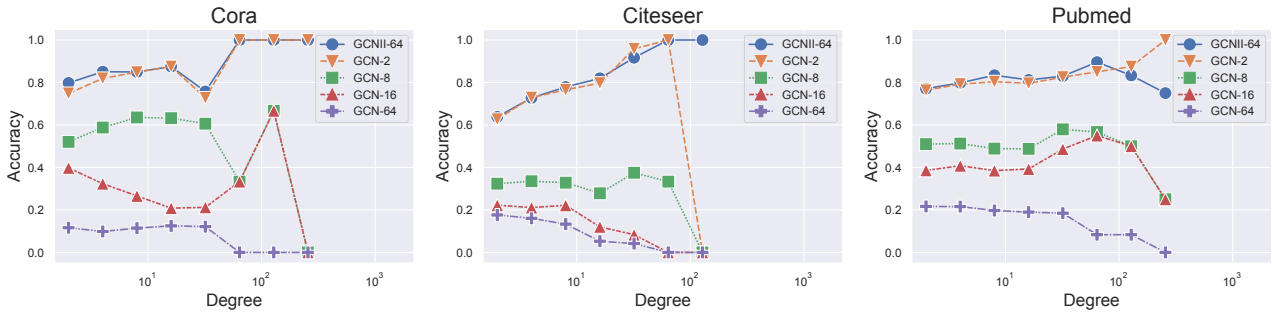

Figure 1. Semi-supervised node classification accuracy v.s. degree.

Figure 2. Ablation study on initial residual and identity mapping.

supported by Alibaba Group through Alibaba Innovative Research Program.

References

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Steeg, G. V., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *ICML*, 2019.

Chen, J., Ma, T., and Xiao, C. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018a.

Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. In *ICML*, 2018b.

Chiang, W., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, pp. 257–266. ACM, 2019.

Chung, F. Four proofs for the cheeger inequality and graph partition algorithms. In *Proceedings of ICCM*, volume 2, pp. 378, 2007.

Dave, V. S., Zhang, B., Chen, P., and Hasan, M. A. Neural-brane: Neural bayesian personalized ranking for attributed network embedding. *Data Science and Engineering*, 4(2):119–131, 2019.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, pp. 3837–3845, 2016.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Fout, A., Byrd, J., Shariat, B., and Ben-Hur, A. Protein interface prediction using graph convolutional networks. In *NeurIPS*, pp. 6530–6539, 2017.

Gao, H. and Ji, S. Graph u-nets. In *ICML*, 2019.

Guo, S., Lin, Y., Feng, N., Song, C., and Wan, H. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *AAAI*, 2019.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

Hardt, M. and Ma, T. Identity matters in deep learning. In *ICLR*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

Huang, W., Zhang, T., Rong, Y., and Huang, J. Adaptive sampling towards fast graph representation learning. In *NeurIPS*, pp. 4563–4572, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019a.

Klicpera, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. In *NeurIPS*, pp. 13333–13345, 2019b.

LeCun, Y., Bengio, Y., et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Lee, J., Lee, I., and Kang, J. Self-attention graph pooling. In *ICML*, 2019.

Li, C. and Goldwasser, D. Encoding social information with graph convolutional networks forpolitical perspective detection in news media. In *ACL*, 2019.

Li, H., Yang, Y., Chen, D., and Lin, Z. Optimization algorithm inspired deep neural network structure design. *arXiv preprint arXiv:1810.01638*, 2018a.

Li, J., Han, Z., Cheng, H., Su, J., Wang, P., Zhang, J., and Pan, L. Predicting path failure in time-evolving graphs. In *KDD*. ACM, 2019.

Li, Q., Han, Z., and Wu, X. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018b.

Li, R., Wang, S., Zhu, F., and Huang, J. Adaptive graph convolutional neural networks. In *AAAI*, 2018c.

Liu, Z., Chen, C., Li, L., Zhou, J., Li, X., Song, L., and Qi, Y. Geniepath: Graph neural networks with adaptive receptive paths. In *AAAI*, 2019.

Ma, J., Wen, J., Zhong, M., Chen, W., and Li, X. MMM: multi-source multi-net micro-video recommendation with clustered hidden item representation learning. *Data Science and Engineering*, 4(3):240–253, 2019.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. In *ICLR*, 2020.

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

Papyan, V., Romano, Y., and Elad, M. Convolutional neural networks analyzed via convolutional sparse coding. *The Journal of Machine Learning Research*, 18(1):2887–2938, 2017.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2020.

Qiu, J., Tang, J., Ma, H., Dong, Y., Wang, K., and Tang, J. Deepinf: Social influence prediction with deep learning. In *KDD*, pp. 2110–2119. ACM, 2018.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-scale attributed node embedding, 2019.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., and Eliassi-Rad, T. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

Shang, J., Xiao, C., Ma, T., Li, H., and Sun, J. Gamenet: Graph augmented memory networks for recommending medication combination. In *AAAI*, 2019.

Thekumparampil, K. K., Wang, C., Oh, S., and Li, L.-J. Attention-based graph neural network for semi-supervised learning, 2018.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. *ICLR*, 2018.

Velickovic, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. Deep graph infomax. In *ICLR*, 2019.

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

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *ICML*, pp. 6861–6871, 2019.

Xu, B., Shen, H., Cao, Q., Qiu, Y., and Cheng, X. Graph wavelet neural network. In *ICLR*, 2019.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pp. 974–983. ACM, 2018.

Zhang, J. and Ghanem, B. Ista-net: Interpretable optimization-inspired deep network for image compressive sensing. In *CVPR*, pp. 1828–1837, 2018.

Zhang, J., Shi, X., Xie, J., Ma, H., King, I., and Yeung, D. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *UAI*, 2018.

Zhao, L., Peng, X., Tian, Y., Kapadia, M., and Metaxas, D. N. Semantic graph convolutional networks for 3d human pose regression. In *CVPR*, pp. 3425–3435, 2019.

A. Proofs

A.1. Proof of Theorem 2

Proof. For simplicity, we assume the signal vector $\mathbf{x}$ to be non-negative. Note that we can convert $\mathbf{x}$ into a non-negative input layer $\mathbf{H}^{(0)}$ by a linear transformation. We consider a weaker version of GCNII by fixing $\alpha_\ell = 0.5$ and fixing the weight matrix $(1-\beta_\ell)\mathbf{I}_n + \beta_\ell \mathbf{W}^{(\ell)}$ to be $\gamma_\ell \mathbf{I}_n$, where γ_ℓ is a learnable parameter. We have

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\left(\mathbf{H}^{(\ell)}+\mathbf{x}\right)\gamma_\ell \mathbf{I}_n\right).$$

Since the input feature $\mathbf{x}$ is non-negative, we can remove the ReLU operation:

$$\begin{aligned}\mathbf{H}^{(\ell+1)} &= \gamma_\ell \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\left(\mathbf{H}^{(\ell)}+\mathbf{x}\right)\\ &= \gamma_\ell\left(\left(\mathbf{I}_n - \tilde{\mathbf{L}}\right)\cdot\left(\mathbf{H}^{(\ell)}+\mathbf{x}\right)\right).\end{aligned}$$

Consequently, we can express the final representation as

$$\mathbf{H}^{(K-1)} = \left(\sum_{\ell=0}^{K-1}\left(\prod_{k=K-\ell-1}^{K-1}\gamma_k\right)\left(\mathbf{I}_n-\tilde{\mathbf{L}}\right)^\ell\right)\mathbf{x}. \quad (8)$$

On the other hand, a polynomial filter of graph $\tilde{G}$ can be expressed as

$$\left(\sum_{k=0}^{K-1}\theta_k\tilde{\mathbf{L}}^k\right)\mathbf{x} = \left(\sum_{i=0}^{k}\theta_k\left(\mathbf{I}_n-\left(\mathbf{I}_n-\tilde{\mathbf{L}}\right)\right)^k\right)\mathbf{x}$$
$$= \left(\sum_{k=0}^{K-1}\theta_k\left(\sum_{\ell=0}^{k}(-1)^\ell\binom{k}{\ell}\left(\mathbf{I}_n-\tilde{\mathbf{L}}\right)^\ell\right)\right)\mathbf{x}.$$

Switching the order of summation follows that a K-order polynomial fiter $\left(\sum_{k=0}^{K-1}\theta_k\tilde{\mathbf{L}}^k\right)\mathbf{x}$ can be expressed as

$$\left(\sum_{k=0}^{K-1}\theta_k\tilde{\mathbf{L}}^k\right)\mathbf{x}=\left(\sum_{\ell=0}^{K-1}\left(\sum_{k=\ell}^{K-1}\theta_k(-1)^\ell\binom{k}{\ell}\right)\left(\mathbf{I}_n-\tilde{\mathbf{L}}\right)^\ell\right)\mathbf{x}. \quad (9)$$

To show that GCNII can express an arbitrary K-order polynomial filter, we need to prove that there exists a solution γ_ℓ, $\ell = 0,\dots,K-1$ such that the corresponding coefficients of $\left(\mathbf{I}_n-\tilde{\mathbf{L}}\right)^\ell$ in equations (8) and (9) are equivalent. More precisely, we need to show the following equation system

$$\prod_{k=K-\ell-1}^{K-1}\gamma_k = \sum_{k=\ell}^{K-1}\theta_k(-1)^\ell\binom{k}{\ell},\ k=0,\dots,K-1,$$

has a solution γ_ℓ, $\ell = 0,\dots,K-1$. Since the left-hand side is a partial product of γ_k from $K-\ell-1$ to $K-1$, we can solve the equation system by

$$\gamma_{K-\ell-1} = \sum_{k=\ell}^{K-1}\theta_k(-1)^\ell\binom{k}{\ell}\Bigg/\sum_{k=\ell-1}^{K-1}\theta_k(-1)^{\ell-1}\binom{k}{\ell-1}, \quad (10)$$

for $\ell = 1,\dots,K-1$ and $\gamma_{K-1} = \sum_{k=0}^{K-1}\theta_k$. Note that the above solution may fail when $\sum_{k=\ell-1}^{K-1}\theta_k(-1)^{\ell-1}\binom{k}{\ell-1} = 0$. In this case, we can set $\gamma_{K-\ell-1}$ sufficiently large so that equation (10) is still a good approximation. We also note that this case is rare because it implies that the K-order filter ignores all features from the ℓ-hop neighbors. This proves that a K-layer GCNII can express the K-th order polynomial filter $\left(\sum_{i=0}^{k}\theta_i\mathbf{L}^i\right)\mathbf{x}$ with arbitrary coefficients θ. □

A.2. Proof of Theorem 1

To prove Theorem 1, we need the following *Cheeger Inequality* (Chung, 2007) for lazy random walks.

Lemma 1 ((Chung, 2007)). *Let $\mathbf{p}_i^{(K)} = \left(\frac{\mathbf{I}_n+\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1}}{2}\right)^K\mathbf{e_i}$ is the K-th transition probability vector from node i on connected self-looped graph $\tilde{G}$. Let $\lambda_{\tilde{G}}$ denote the spectral gap of $\tilde{G}$. The j-th entry of $\mathbf{p}_i^{(K)}$ can be bounded by*

$$\left|\mathbf{p}_i^{(K)}(j) - \frac{d_j+1}{2m+n}\right| \le \sqrt{\frac{d_j+1}{d_i+1}}\left(1-\frac{\lambda_{\tilde{G}}^2}{2}\right)^K.$$

Proof of Theorem 1. Note that $\mathbf{I}_n = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{D}}^{1/2}$, we have

$$\begin{aligned}\mathbf{h}^{(K)} &= \left(\frac{\mathbf{I}_n+\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}}{2}\right)^K\cdot\mathbf{x}\\ &= \left(\tilde{\mathbf{D}}^{-1/2}\left(\frac{\mathbf{I}_n+\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1}}{2}\right)\tilde{\mathbf{D}}^{1/2}\right)^K\cdot\mathbf{x}\\ &= \tilde{\mathbf{D}}^{-1/2}\left(\frac{\mathbf{I}_n+\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1}}{2}\right)^K\cdot\left(\tilde{\mathbf{D}}^{1/2}\mathbf{x}\right).\end{aligned}$$

We express $\tilde{\mathbf{D}}^{1/2}\mathbf{x}$ as linear combination of standard basis:

$$\tilde{\mathbf{D}}^{1/2}\mathbf{x} = (\mathbf{D}+\mathbf{I}_n)^{1/2}\mathbf{x} = \sum_{i=1}^{n}\left(\mathbf{x}(i)\sqrt{d_i+1}\right)\cdot\mathbf{e_i},$$

it follows that

$$\begin{aligned}\mathbf{h}^{(K)} &= \tilde{\mathbf{D}}^{-1/2}\left(\frac{\mathbf{I}_n+\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1}}{2}\right)^K\cdot\sum_{i=1}^{n}\left(\mathbf{x}(i)\sqrt{d_i+1}\right)\cdot\mathbf{e_i}\\ &= \sum_{i=1}^{n}\mathbf{x}(i)\sqrt{d_i+1}\cdot\tilde{\mathbf{D}}^{-1/2}\left(\frac{\mathbf{I}_n+\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1}}{2}\right)^K\cdot\mathbf{e_i}.\end{aligned}$$

We note that $\left(\frac{\mathbf{I}_n+\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1}}{2}\right)^K \cdot \mathbf{e_i} = \mathbf{p}_i^{(K)}$ is the K-th transition probability vector of a random walk from node i. By Lemma 1, the j-th entry of $\mathbf{p}_i^{(K)}$ can be bounded by

$$\left|\mathbf{p}_i^{(K)}(j) - \frac{d_j+1}{2m+n}\right| \leq \sqrt{\frac{d_j+1}{d_i+1}}\left(1-\frac{\lambda_{\tilde{G}}^2}{2}\right)^K,$$

or equivalently,

$$\mathbf{p}_i^{(K)}(j) = \frac{d_j+1}{2m+n} \pm \sqrt{\frac{d_j+1}{d_i+1}}\left(1-\frac{\lambda_{\tilde{G}}^2}{2}\right)^K.$$

Therefore, we can express the j-th entry of $\mathbf{h}^{(K)}$ as

$$\mathbf{h}^{(K)}(j) = \left(\sum_{i=1}^n \sqrt{d_i+1}\mathbf{x}(i)\cdot\tilde{\mathbf{D}}^{-1/2}\mathbf{p}_i^{(K)}\right)(j)$$

$$=\sum_{i=1}^n \sqrt{d_i+1}\mathbf{x}(i)\frac{1}{\sqrt{d_j+1}}\cdot\left(\frac{d_j+1}{2m+n}\pm\sqrt{\frac{d_j+1}{d_i+1}}\left(1-\frac{\lambda_{\tilde{G}}^2}{2}\right)^K\right)$$

$$=\sum_{i=1}^n \frac{\sqrt{(d_j+1)(d_i+1)}}{2m+n}\mathbf{x}(i) \pm \sum_{i=1}^n \mathbf{x}(i)\left(1-\frac{\lambda_{\tilde{G}}^2}{2}\right)^K.$$

This proves

$$\mathbf{h}^{(K)} = \frac{\left\langle\tilde{\mathbf{D}}^{1/2}\mathbf{1},\mathbf{x}\right\rangle}{2m+n}\tilde{\mathbf{D}}^{1/2}\mathbf{1} \pm \left(\sum_{i=1}^n x_i\right)\cdot\left(1-\frac{\lambda_{\tilde{G}}^2}{2}\right)^K\cdot\mathbf{1},$$

and the Theorem follows. □

B. Hyper-parameters details

Table 6 summarizes the training configuration of GCNII for semi-supervised. L_{2_d} and L_{2_c} denote the weight decay for dense layer and convolutional layer respectively. The searching hyper-parameters include numbers of layers, hidden dimension, dropout, λ and L_{2_c} regularization.

Table 7 summarizes the training configuration of all model for full-supervised. We use the full-supervised hyper-parameter setting from DropEdge for JKNet and IncepGCN on citation networks. For other cases, grid search was performed over the following search space: layers (4, 8, 16, 32 ,64), dropedge (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9), α_ℓ (0.1, 0.2, 0.3, 0.4, 0.5), λ (0.5, 1, 1.5), L_2 regularization (1e-3, 5e-4, 1e-4, 5e-5, 1e-5, 5e-6, 1e-6).

Table 6. The hyper-parameters for Table 2.

Dataset	Hyper-parameters
Cora	layers: 64, α_ℓ: 0.1, lr: 0.01, hidden: 64, λ: 0.5, dropout: 0.6, L_{2_c}: 0.01, L_{2_d}: 0.0005
Citeseer	layers: 32, α_ℓ: 0.1, lr: 0.01, hidden: 256, λ: 0.6, dropout: 0.7, L_{2_c}: 0.01, L_{2_d}: 0.0005
Pubmed	layers: 16, α_ℓ: 0.1, lr: 0.01, hidden: 256, λ: 0.4, dropout: 0.5, L_{2_c}: 0.0005, L_{2_d}: 0.0005

Table 7. The hyper-parameters for Table 5.

Dataset	Method	Hyper-parameters
Cora	APPNP	α: 0.1, L_2: 0.0005, lr: 0.01, hidden: 64, dropout: 0.5
	GCNII	layers: 64, α_ℓ: 0.2, lr: 0.01, hidden: 64, λ: 0.5, dropout: 0.5, L_2: 0.0001
Cite.	APPNP	α: 0.5, L_2: 0.0005, lr: 0.01, hidden: 64, dropout: 0.5
	GCNII	layers: 64, α_ℓ: 0.5, lr: 0.01, hidden: 64, λ: 0.5, dropout: 0.5, L_2: 5e-6
Pubm.	APPNP	α: 0.4, L_2: 0.0001, lr: 0.01, hidden: 64, dropout: 0.5
	GCNII	layers: 64, α_ℓ: 0.1, lr: 0.01, hidden: 64, λ: 0.5, dropout: 0.5, L_2: 5e-6
Cham.	APPNP	α: 0.1, L_2: 1e-6, lr: 0.01, hidden: 64, dropout: 0.5
	JKNet	layers: 32, lr: 0.01, hidden: 64, dropedge: 0.7, dropout: 0.5, L_2: 0.0001
	IncepGCN	layers: 8, lr: 0.01, hidden: 64, dropedge: 0.9, dropout: 0.5, L_2: 0.0005
	GCNII	layers: 8, α_ℓ: 0.2, lr: 0.01, hidden: 64, λ: 1.5, dropout: 0.5, L_2: 0.0005
Corn.	APPNP	α: 0.5, L_2: 0.005, lr: 0.01, hidden: 64, dropout: 0.5
	JKNet	layers: 4, lr: 0.01, hidden: 64, dropedge: 0.5, dropout: 0.5, L_2: 5e-5
	IncepGCN	layers: 16, lr: 0.01, hidden: 64, dropedge: 0.7, dropout: 0.5, L_2: 5e-5
	GCNII	layers: 16, α_ℓ: 0.5, lr: 0.01, hidden: 64, λ: 1, dropout: 0.5, L_2: 0.001
Texa.	APPNP	α: 0.5, L_2: 0.001, lr: 0.01, hidden: 64, dropout: 0.5
	JKNet	layers: 32, lr: 0.01, hidden: 64, dropedge: 0.8, dropout: 0.5, L_2: 5e-5
	IncepGCN	layers: 8, lr: 0.01, hidden: 64, dropedge: 0.8, dropout: 0.5, L_2: 5e-6
	GCNII	layers: 32, α_ℓ: 0.5, lr: 0.01, hidden: 64, λ: 1.5, dropout: 0.5, L_2: 0.0001
Wisc.	APPNP	α: 0.5, L_2: 0.005, lr: 0.01, hidden: 64, dropout: 0.5
	JKNet	layers: 8, lr: 0.01, hidden: 64, dropedge: 0.8, dropout: 0.5, L_2: 5e-5
	IncepGCN	layers: 8, lr: 0.01, hidden: 64, dropedge: 0.7, dropout: 0.5, L_2: 0.0001
	GCNII	layers: 16, α_ℓ: 0.5, lr: 0.01, hidden: 64, λ: 1, dropout: 0.5, L_2: 0.0005