Simplifying Graph Convolutional Networks

Felix Wu [*1] **Tianyi Zhang** [*1] **Amauri Holanda de Souza Jr.** [*1 2] **Christopher Fifty** [1] **Tao Yu** [1]
Kilian Q. Weinberger [1]

Abstract

Graph Convolutional Networks (GCNs) and their variants have experienced significant attention and have become the de facto methods for learning graph representations. GCNs derive inspiration primarily from recent deep learning approaches, and as a result, may inherit unnecessary complexity and redundant computation. In this paper, we reduce this excess complexity through successively removing nonlinearities and collapsing weight matrices between consecutive layers. We theoretically analyze the resulting linear model and show that it corresponds to a fixed low-pass filter followed by a linear classifier. Notably, our experimental evaluation demonstrates that these simplifications do not negatively impact accuracy in many downstream applications. Moreover, the resulting model scales to larger datasets, is naturally interpretable, and yields up to two orders of magnitude speedup over FastGCN.

1. Introduction

Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017) are an efficient variant of Convolutional Neural Networks (CNNs) on graphs. GCNs stack layers of learned first-order spectral filters followed by a nonlinear activation function to learn graph representations. Recently, GCNs and subsequent variants have achieved state-of-the-art results in various application areas, including but not limited to citation networks (Kipf & Welling, 2017), social networks (Chen et al., 2018), applied chemistry (Liao et al., 2019), natural language processing (Yao et al., 2019; Han et al., 2012; Zhang et al., 2018c), and computer vision (Wang et al., 2018; Kampffmeyer et al., 2018).

Historically, the development of machine learning algorithms has followed a clear trend from initial simplicity to need-driven complexity. For instance, limitations of the linear Perceptron (Rosenblatt, 1958) motivated the development of the more complex but also more expressive neural network (or multi-layer Perceptrons, MLPs) (Rosenblatt, 1961). Similarly, simple pre-defined linear image filters (Sobel & Feldman, 1968; Harris & Stephens, 1988) eventually gave rise to nonlinear CNNs with learned convolutional kernels (Waibel et al., 1989; LeCun et al., 1989). As additional algorithmic complexity tends to complicate theoretical analysis and obfuscates understanding, it is typically only introduced for applications where simpler methods are insufficient. Arguably, most classifiers in real world applications are still linear (typically logistic regression), which are straight-forward to optimize and easy to interpret.

However, possibly because GCNs were proposed after the recent "renaissance" of neural networks, they tend to be a rare exception to this trend. GCNs are built upon multi-layer neural networks, and were never an extension of a simpler (insufficient) linear counterpart.

In this paper, we observe that GCNs inherit considerable complexity from their deep learning lineage, which can be burdensome and unnecessary for less demanding applications. Motivated by the glaring historic omission of a simpler predecessor, we aim to derive the simplest linear model that "could have" preceded the GCN, had a more "traditional" path been taken. We reduce the excess complexity of GCNs by repeatedly removing the nonlinearities between GCN layers and collapsing the resulting function into a single linear transformation. We empirically show that the final linear model exhibits comparable or even superior performance to GCNs on a variety of tasks while being computationally more efficient and fitting significantly fewer parameters. We refer to this simplified linear model as Simple Graph Convolution (SGC).

In contrast to its nonlinear counterparts, the SGC is intuitively interpretable and we provide a theoretical analysis from the graph convolution perspective. Notably, feature extraction in SGC corresponds to a single fixed filter applied to each feature dimension. Kipf & Welling (2017) empirically observe that the "renormalization trick", i.e. adding self-loops to the graph, improves accuracy, and we demon-

[*]Equal contribution [1]Cornell University [2]Federal Institute of Ceara (Brazil). Correspondence to: Felix Wu <fw245@cornell.edu>, Tianyi Zhang <tz58@cornell.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

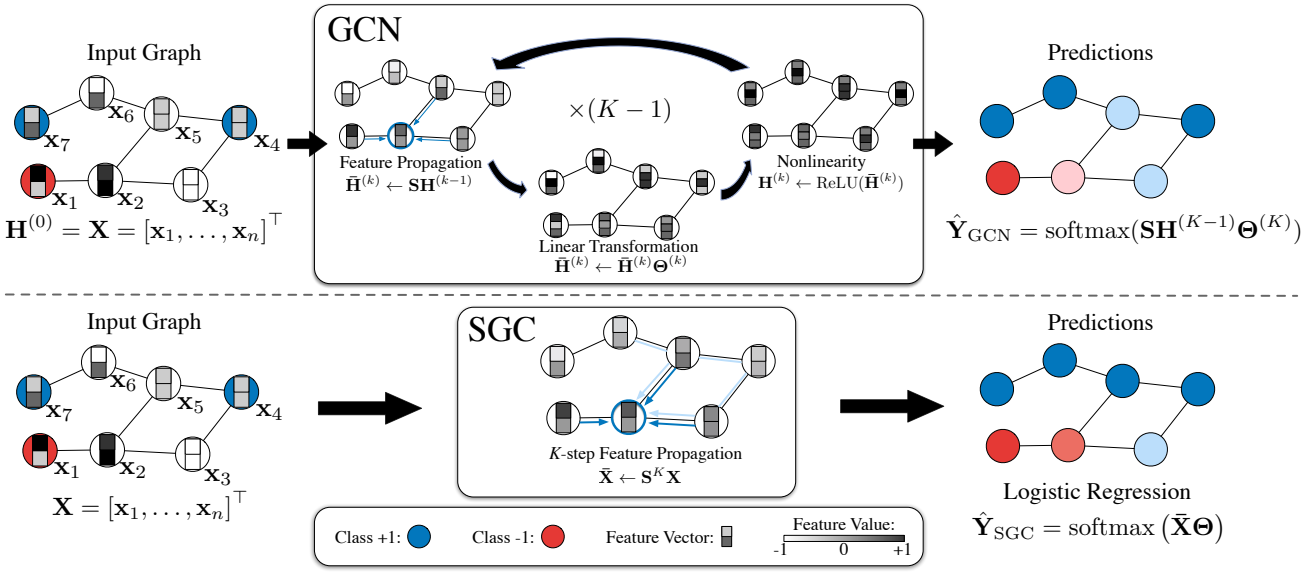

Figure 1. Schematic layout of a GCN v.s. a SGC. *Top row:* The GCN transforms the feature vectors repeatedly throughout K layers and then applies a linear classifier on the final representation. *Bottom row:* the SGC reduces the entire procedure to a simple feature propagation step followed by standard logistic regression.

strate that this method effectively shrinks the graph spectral domain, resulting in a low-pass-type filter when applied to SGC. Crucially, this filtering operation gives rise to locally smooth features across the graph (Bruna et al., 2014).

Through an empirical assessment on node classification benchmark datasets for citation and social networks, we show that the SGC achieves comparable performance to GCN and other state-of-the-art graph neural networks. However, it is significantly faster, and even outperforms FastGCN (Chen et al., 2018) by up to two orders of magnitude on the largest dataset (Reddit) in our evaluation. Finally, we demonstrate that SGC extrapolates its effectiveness to a wide-range of downstream tasks. In particular, SGC rivals, if not surpasses, GCN-based approaches on text classification, user geolocation, relation extraction, and zero-shot image classification tasks. The code is available on Github[1].

2. Simple Graph Convolution

We follow Kipf & Welling (2017) to introduce GCNs (and subsequently SGC) in the context of node classification. Here, GCNs take a graph with some labeled nodes as input and generate label predictions for all graph nodes. Let us formally define such a graph as $\mathcal{G} = (\mathcal{V}, \mathbf{A})$, where $\mathcal{V}$ represents the vertex set consisting of nodes $\{v_1, \ldots, v_n\}$, and $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a symmetric (typically sparse) adjacency matrix where a_{ij} denotes the edge weight between nodes v_i and v_j. A missing edge is represented through $a_{ij} = 0$. We define the degree matrix $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ as a diagonal matrix where each entry on the diagonal is equal to the row-sum of the adjacency matrix $d_i = \sum_j a_{ij}$.

Each node v_i in the graph has a corresponding d-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^d$. The entire feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ stacks n feature vectors on top of one another, $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$. Each node belongs to one out of C classes and can be labeled with a C-dimensional one-hot vector $\mathbf{y}_i \in \{0, 1\}^C$. We only know the labels of a subset of the nodes and want to predict the unknown labels.

2.1. Graph Convolutional Networks

Similar to CNNs or MLPs, GCNs learn a new feature representation for the feature $\mathbf{x}_i$ of each node over multiple layers, which is subsequently used as input into a linear classifier. For the k-th graph convolution layer, we denote the input node representations of all nodes by the matrix $\mathbf{H}^{(k-1)}$ and the output node representations $\mathbf{H}^{(k)}$. Naturally, the initial node representations are just the original input features:

$$\mathbf{H}^{(0)} = \mathbf{X}, \tag{1}$$

which serve as input to the first GCN layer.

A K-layer GCN is identical to applying a K-layer MLP to the feature vector $\mathbf{x}_i$ of each node in the graph, except that the hidden representation of each node is averaged with its neighbors at the beginning of each layer. In each graph convolution layer, node representations are updated in three

[1] https://github.com/Tiiiger/SGC

stages: feature propagation, linear transformation, and a pointwise nonlinear activation (see Figure 1). For the sake of clarity, we describe each step in detail.

Feature propagation is what distinguishes a GCN from an MLP. At the beginning of each layer the features $\mathbf{h}_i$ of each node v_i are averaged with the feature vectors in its local neighborhood,

$$\bar{\mathbf{h}}_i^{(k)} \leftarrow \frac{1}{d_i+1}\mathbf{h}_i^{(k-1)} + \sum_{j=1}^{n} \frac{a_{ij}}{\sqrt{(d_i+1)(d_j+1)}}\mathbf{h}_j^{(k-1)}. \tag{2}$$

More compactly, we can express this update over the entire graph as a simple matrix operation. Let $\mathbf{S}$ denote the "normalized" adjacency matrix with added self-loops,

$$\mathbf{S} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}, \tag{3}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$. The simultaneous update in Equation 2 for all nodes becomes a simple sparse matrix multiplication

$$\bar{\mathbf{H}}^{(k)} \leftarrow \mathbf{S}\mathbf{H}^{(k-1)}. \tag{4}$$

Intuitively, this step smoothes the hidden representations locally along the edges of the graph and ultimately encourages similar predictions among locally connected nodes.

Feature transformation and nonlinear transition. After the local smoothing, a GCN layer is identical to a standard MLP. Each layer is associated with a learned weight matrix $\mathbf{\Theta}^{(k)}$, and the smoothed hidden feature representations are transformed linearly. Finally, a nonlinear activation function such as ReLU is applied pointwise before outputting feature representation $\mathbf{H}^{(k)}$. In summary, the representation updating rule of the k-th layer is:

$$\mathbf{H}^{(k)} \leftarrow \mathrm{ReLU}\left(\bar{\mathbf{H}}^{(k)}\mathbf{\Theta}^{(k)}\right). \tag{5}$$

The pointwise nonlinear transformation of the k-th layer is followed by the feature propagation of the $(k+1)$-th layer.

Classifier. For node classification, and similar to a standard MLP, the last layer of a GCN predicts the labels using a *softmax* classifier. Denote the class predictions for n nodes as $\hat{\mathbf{Y}} \in \mathbb{R}^{n\times C}$ where $\hat{y}_{ic}$ denotes the probability of node i belongs to class c. The class prediction $\hat{\mathbf{Y}}$ of a K-layer GCN can be written as:

$$\hat{\mathbf{Y}}_{\mathrm{GCN}} = \mathrm{softmax}\left(\mathbf{S}\mathbf{H}^{(K-1)}\mathbf{\Theta}^{(K)}\right), \tag{6}$$

where $\mathrm{softmax}(\mathbf{x}) = \exp(\mathbf{x})/\sum_{c=1}^{C}\exp(x_c)$ acts as a normalizer across all classes.

2.2. Simple Graph Convolution

In a traditional MLP, deeper layers increase the expressivity because it allows the creation of feature hierarchies, e.g. features in the second layer build on top of the features of the first layer. In GCNs, the layers have a second important function: in each layer the hidden representations are averaged among neighbors that are one hop away. This implies that after k layers a node obtains feature information from all nodes that are $k-$hops away in the graph. This effect is similar to convolutional neural networks, where depth increases the receptive field of internal features (Hariharan et al., 2015). Although convolutional networks can benefit substantially from increased depth (Huang et al., 2016), typically MLPs obtain little benefit beyond 3 or 4 layers.

Linearization. We hypothesize that the nonlinearity between GCN layers is not critical - but that the majority of the benefit arises from the local averaging. We therefore remove the nonlinear transition functions between each layer and only keep the final softmax (in order to obtain probabilistic outputs). The resulting model is linear, but still has the same increased "receptive field" of a K-layer GCN,

$$\hat{\mathbf{Y}} = \mathrm{softmax}\left(\mathbf{S}\ldots\mathbf{S}\mathbf{S}\mathbf{X}\mathbf{\Theta}^{(1)}\mathbf{\Theta}^{(2)}\ldots\mathbf{\Theta}^{(K)}\right). \tag{7}$$

To simplify notation we can collapse the repeated multiplication with the normalized adjacency matrix $\mathbf{S}$ into a single matrix by raising $\mathbf{S}$ to the K-th power, $\mathbf{S}^K$. Further, we can reparameterize our weights into a single matrix $\mathbf{\Theta} = \mathbf{\Theta}^{(1)}\mathbf{\Theta}^{(2)}\ldots\mathbf{\Theta}^{(K)}$. The resulting classifier becomes

$$\hat{\mathbf{Y}}_{\mathrm{SGC}} = \mathrm{softmax}\left(\mathbf{S}^K\mathbf{X}\mathbf{\Theta}\right), \tag{8}$$

which we refer to as Simple Graph Convolution (SGC).

Logistic regression. Equation 8 gives rise to a natural and intuitive interpretation of SGC: by distinguishing between feature extraction and classifier, SGC consists of a fixed (i.e., parameter-free) feature extraction/smoothing component $\bar{\mathbf{X}} = \mathbf{S}^K\mathbf{X}$ followed by a linear logistic regression classifier $\hat{\mathbf{Y}} = \mathrm{softmax}(\bar{\mathbf{X}}\mathbf{\Theta})$. Since the computation of $\bar{\mathbf{X}}$ requires no weight it is essentially equivalent to a feature pre-processing step and the entire training of the model reduces to straight-forward multi-class logistic regression on the pre-processed features $\bar{\mathbf{X}}$.

Optimization details. The training of logistic regression is a well studied convex optimization problem and can be performed with any efficient second order method or stochastic gradient descent (Bottou, 2010). Provided the graph connectivity pattern is sufficiently sparse, SGD naturally scales to very large graph sizes and the training of SGC is drastically faster than that of GCNs.

3. Spectral Analysis

We now study SGC from a graph convolution perspective. We demonstrate that SGC corresponds to a fixed filter on the graph spectral domain. In addition, we show that adding self-loops to the original graph, i.e. the renormalization trick (Kipf & Welling, 2017), effectively shrinks the underlying graph spectrum. On this scaled domain, SGC acts as a low-pass filter that produces smooth features over the graph. As a result, nearby nodes tend to share similar representations and consequently predictions.

3.1. Preliminaries on Graph Convolutions

Analogous to the Euclidean domain, graph Fourier analysis relies on the spectral decomposition of graph Laplacians. The graph Laplacian $\mathbf{\Delta} = \mathbf{D} - \mathbf{A}$ (as well as its normalized version $\mathbf{\Delta}_{\text{sym}} = \mathbf{D}^{-1/2}\mathbf{\Delta}\mathbf{D}^{-1/2}$) is a symmetric positive semidefinite matrix with eigendecomposition $\mathbf{\Delta} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, where $\mathbf{U} \in \mathbb{R}^{n\times n}$ comprises orthonormal eigenvectors and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_n)$ is a diagonal matrix of eigenvalues. The eigendecomposition of the Laplacian allows us to define the Fourier transform equivalent on the graph domain, where eigenvectors denote Fourier modes and eigenvalues denote frequencies of the graph. In this regard, let $\mathbf{x} \in \mathbb{R}^n$ be a signal defined on the vertices of the graph. We define the graph Fourier transform of $\mathbf{x}$ as $\hat{\mathbf{x}} = \mathbf{U}^\top\mathbf{x}$, with inverse operation given by $\mathbf{x} = \mathbf{U}\hat{\mathbf{x}}$. Thus, the graph convolution operation between signal $\mathbf{x}$ and filter $\mathbf{g}$ is

$$\mathbf{g} * \mathbf{x} = \mathbf{U}\left((\mathbf{U}^\top\mathbf{g}) \odot (\mathbf{U}^\top\mathbf{x})\right) = \mathbf{U}\hat{\mathbf{G}}\mathbf{U}^\top\mathbf{x}, \quad (9)$$

where $\hat{\mathbf{G}} = \text{diag}\left(\hat{g}_1, \dots, \hat{g}_n\right)$ denotes a diagonal matrix in which the diagonal corresponds to spectral filter coefficients.

Graph convolutions can be approximated by k-th order polynomials of Laplacians

$$\mathbf{U}\hat{\mathbf{G}}\mathbf{U}^\top\mathbf{x} \approx \sum_{i=0}^{k}\theta_i\mathbf{\Delta}^i\mathbf{x} = \mathbf{U}\left(\sum_{i=0}^{k}\theta_i\mathbf{\Lambda}^i\right)\mathbf{U}^\top\mathbf{x}, \quad (10)$$

where θ_i denotes coefficients. In this case, filter coefficients correspond to polynomials of the Laplacian eigenvalues, i.e., $\hat{\mathbf{G}} = \sum_i \theta_i\mathbf{\Lambda}^i$ or equivalently $\hat{g}(\lambda_j) = \sum_i \theta_i\lambda_j^i$.

Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017) employ an affine approximation ($k = 1$) of Equation 10 with coefficients $\theta_0 = 2\theta$ and $\theta_1 = -\theta$ from which we attain the basic GCN convolution operation

$$\mathbf{g} * \mathbf{x} = \theta(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x}. \quad (11)$$

In their final design, Kipf & Welling (2017) replace the matrix $\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ by a normalized version $\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and consequently $\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$, dubbed the *renormalization trick*. Finally, by generalizing the convolution to work with multiple filters in a d-channel input and layering the model with nonlinear activation functions between each layer, we have the GCN propagation rule as defined in Equation 5.

3.2. SGC and Low-Pass Filtering

The initial first-order Chebyshev filter derived in GCNs corresponds to the propagation matrix $\mathbf{S}_{\text{1-order}} = \mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ (see Equation 11). Since the normalized Laplacian is $\mathbf{\Delta}_{\text{sym}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, then $\mathbf{S}_{\text{1-order}} = 2\mathbf{I} - \mathbf{\Delta}_{\text{sym}}$. Therefore, feature propagation with $\mathbf{S}_{\text{1-order}}^K$ implies filter coefficients $\hat{g}_i = \hat{g}(\lambda_i) = (2 - \lambda_i)^K$, where λ_i denotes the eigenvalues of $\mathbf{\Delta}_{\text{sym}}$. Figure 2 illustrates the filtering operation related to $\mathbf{S}_{\text{1-order}}$ for a varying number of propagation steps $K \in \{1, \dots, 6\}$. As one may observe, high powers of $\mathbf{S}_{\text{1-order}}$ lead to exploding filter coefficients and undesirably over-amplify signals at frequencies $\lambda_i < 1$.

To tackle potential numerical issues associated with the first-order Chebyshev filter, Kipf & Welling (2017) propose the *renormalization trick*. Basically, it consists of replacing $\mathbf{S}_{\text{1-order}}$ by the normalized adjacency matrix after adding self-loops for all nodes. We call the resulting propagation matrix the augmented normalized adjacency matrix $\tilde{\mathbf{S}}_{\text{adj}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$, where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$. Correspondingly, we define the augmented normalized Laplacian $\tilde{\mathbf{\Delta}}_{\text{sym}} = \mathbf{I} - \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$. Thus, we can describe the spectral filters associated with $\tilde{\mathbf{S}}_{\text{adj}}$ as a polynomial of the eigenvalues of the underlying Laplacian, i.e., $\hat{g}(\tilde{\lambda}_i) = (1 - \tilde{\lambda}_i)^K$, where $\tilde{\lambda}_i$ are eigenvalues of $\tilde{\mathbf{\Delta}}_{\text{sym}}$.

We now analyze the spectrum of $\tilde{\mathbf{\Delta}}_{\text{sym}}$ and show that adding self-loops to graphs shrinks the spectrum (eigenvalues) of the corresponding normalized Laplacian.

Theorem 1. *Let $\mathbf{A}$ be the adjacency matrix of an undirected, weighted, simple graph $\mathcal{G}$ without isolated nodes and with corresponding degree matrix $\mathbf{D}$. Let $\tilde{\mathbf{A}} = \mathbf{A} + \gamma\mathbf{I}$, such that $\gamma > 0$, be the augmented adjacency matrix with corresponding degree matrix $\tilde{\mathbf{D}}$. Also, let λ_1 and λ_n denote the smallest and largest eigenvalues of $\mathbf{\Delta}_{sym} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$; similarly, let $\tilde{\lambda}_1$ and $\tilde{\lambda}_n$ be the smallest and largest eigenvalues of $\tilde{\mathbf{\Delta}}_{sym} = \mathbf{I} - \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$. We have that*

$$0 = \lambda_1 = \tilde{\lambda}_1 < \tilde{\lambda}_n < \lambda_n. \quad (12)$$

Theorem 1 shows that the largest eigenvalue of the normalized graph Laplacian becomes smaller after adding self-loops $\gamma > 0$ (see supplementary materials for the proof).

Figure 2 depicts the filtering operations associated with the normalized adjacency $\mathbf{S}_{\text{adj}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ and its augmented variant $\tilde{\mathbf{S}}_{\text{adj}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ on the Cora dataset (Sen et al., 2008). Feature propagation with $\mathbf{S}_{\text{adj}}$ corresponds to filters $g(\lambda_i) = (1 - \lambda_i)^K$ in the spectral range

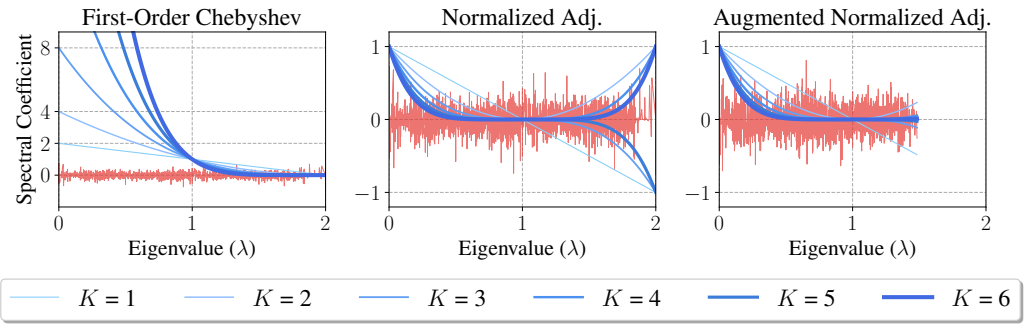

Figure 2. Feature (red) and filters (blue) spectral coefficients for different propagation matrices on Cora dataset (3rd feature).

$[0, 2]$; therefore odd powers of $\mathbf{S}_{\text{adj}}$ yield negative filter coefficients at frequencies $\lambda_i > 1$. By adding self-loops ($\tilde{\mathbf{S}}_{\text{adj}}$), the largest eigenvalue shrinks from 2 to approximately 1.5 and then eliminates the effect of negative coefficients. Moreover, this scaled spectrum allows the filter defined by taking powers $K > 1$ of $\tilde{\mathbf{S}}_{\text{adj}}$ to act as a low-pass-type filters. In supplementary material, we empirically evaluate different choices for the propagation matrix.

4. Related Works

4.1. Graph Neural Networks

Bruna et al. (2014) first propose a spectral graph-based extension of convolutional networks to graphs. In a follow-up work, ChebyNets (Defferrard et al., 2016) define graph convolutions using Chebyshev polynomials to remove the computationally expensive Laplacian eigendecomposition. GCNs (Kipf & Welling, 2017) further simplify graph convolutions by stacking layers of first-order Chebyshev polynomial filters with a redefined propagation matrix $\mathbf{S}$. Chen et al. (2018) propose an efficient variant of GCN based on importance sampling, and Hamilton et al. (2017) propose a framework based on sampling and aggregation. Atwood & Towsley (2016), Abu-El-Haija et al. (2018), and Liao et al. (2019) exploit multi-scale information by raising $\mathbf{S}$ to higher order. Xu et al. (2019) study the expressiveness of graph neural networks in terms of their ability to distinguish any two graphs and introduce Graph Isomorphism Network, which is proved to be as powerful as the Weisfeiler-Lehman test for graph isomorphism. Klicpera et al. (2019) separate the non-linear transformation from propagation by using a neural network followed by a personalized random walk. There are many other graph neural models (Monti et al., 2017; Duran & Niepert, 2017; Li et al., 2018); we refer to Zhou et al. (2018); Battaglia et al. (2018); Wu et al. (2019) for a more comprehensive review.

Previous publications have pointed out that simpler, sometimes linear models can be effective for node/graph classification tasks. Thekumparampil et al. (2018) empirically show that a linear version of GCN can perform competitively and propose an attention-based GCN variant. Cai & Wang (2018) propose an effective linear baseline for graph classification using node degree statistics. Eliav & Cohen (2018) show that models which use linear feature/label propagation steps can benefit from self-training strategies. Li et al. (2019) propose a generalized version of label propagation and provide a similar spectral analysis of the renormalization trick.

Graph Attentional Models learn to assign different edge weights at each layer based on node features and have achieved state-of-the-art results on several graph learning tasks (Velickovic et al., 2018; Thekumparampil et al., 2018; Zhang et al., 2018a; Kampffmeyer et al., 2018). However, the attention mechanism usually adds significant overhead to computation and memory usage. We refer the readers to Lee et al. (2018) for further comparison.

4.2. Other Works on Graphs

Graph methodologies can roughly be categorized into two approaches: graph embedding methods and graph laplacian regularization methods. Graph embedding methods (Weston et al., 2008; Perozzi et al., 2014; Yang et al., 2016; Velikovi et al., 2019) represent nodes as high-dimensional feature vectors. Among them, DeepWalk (Perozzi et al., 2014) and Deep Graph Infomax (DGI) (Velikovi et al., 2019) use unsupervised strategies to learn graph embeddings. DeepWalk relies on truncated random walk and uses a skip-gram model to generate embeddings, whereas DGI trains a graph convolutional encoder through maximizing mutual information. Graph Laplacian regularization (Zhu et al., 2003; Zhou et al., 2004; Belkin & Niyogi, 2004; Belkin et al., 2006) introduce a regularization term based on graph structure which forces nodes to have similar labels to their neighbors. Label Propagation (Zhu et al., 2003) makes predictions by spreading label information from labeled nodes to their neighbors until convergence.

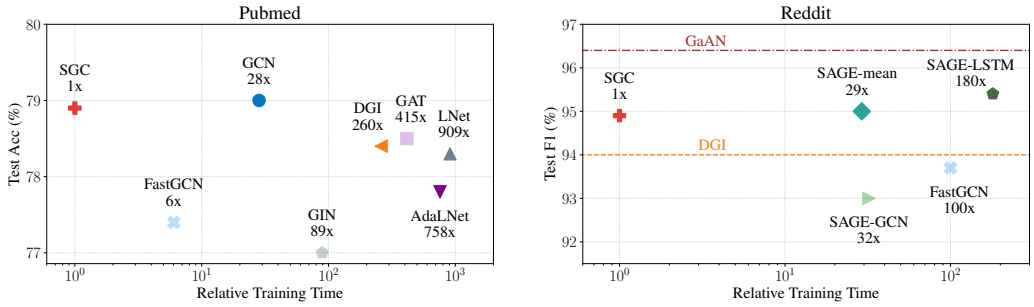

Figure 3. Performance over training time on Pubmed and Reddit. SGC is the fastest while achieving competitive performance. We are not able to benchmark the training time of GaAN and DGI on Reddit because the implementations are not released.

Table 1. Dataset statistics of the citation networks and Reddit.

Dataset	# Nodes	# Edges	Train/Dev/Test Nodes
Cora	2,708	5,429	140/500/1,000
Citeseer	3,327	4,732	120/500/1,000
Pubmed	19,717	44,338	60/500/1,000
Reddit	233K	11.6M	152K/24K/55K

5. Experiments and Discussion

We first evaluate SGC on citation networks and social networks and then extend our empirical analysis to a wide range of downstream tasks.

5.1. Citation Networks & Social Networks

We evaluate the semi-supervised node classification performance of SGC on the Cora, Citeseer, and Pubmed citation network datasets (Table 2) (Sen et al., 2008). We supplement our citation network analysis by using SGC to inductively predict community structure on Reddit (Table 3), which consists of a much larger graph. Dataset statistics are summarized in Table 1.

Datasets and experimental setup. On the citation networks, we train SGC for 100 epochs using Adam (Kingma & Ba, 2015) with learning rate 0.2. In addition, we use weight decay and tune this hyperparameter on each dataset using hyperopt (Bergstra et al., 2015) for 60 iterations on the public split validation set. Experiments on citation networks are conducted *transductively*. On the Reddit dataset, we train SGC with L-BFGS (Liu & Nocedal, 1989) using no regularization, and remarkably, training converges in 2 steps. We evaluate SGC *inductively* by following Chen et al. (2018): we train SGC on a subgraph comprising only training nodes and test with the original graph. On all datasets, we tune the number of epochs based on both convergence behavior and validation accuracy.

Table 2. Test accuracy (%) averaged over 10 runs on citation networks. †We remove the outliers (accuracy $< 75/65/75\%$) when calculating their statistics due to high variance.

	Cora	Citeseer	Pubmed
Numbers from literature:			
GCN	81.5	70.3	79.0
GAT	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
GLN	81.2 ± 0.1	70.9 ± 0.1	78.9 ± 0.1
AGNN	83.1 ± 0.1	71.7 ± 0.1	79.9 ± 0.1
LNet	79.5 ± 1.8	66.2 ± 1.9	78.3 ± 0.3
AdaLNet	80.4 ± 1.1	68.7 ± 1.0	78.1 ± 0.4
DeepWalk	70.7 ± 0.6	51.4 ± 0.5	76.8 ± 0.6
DGI	82.3 ± 0.6	71.8 ± 0.7	76.8 ± 0.6
Our experiments:			
GCN	81.4 ± 0.4	70.9 ± 0.5	79.0 ± 0.4
GAT	83.3 ± 0.7	72.6 ± 0.6	78.5 ± 0.3
FastGCN	79.8 ± 0.3	68.8 ± 0.6	77.4 ± 0.3
GIN	77.6 ± 1.1	66.1 ± 0.9	77.0 ± 1.2
LNet	$80.2 \pm 3.0^{\dagger}$	67.3 ± 0.5	$78.3 \pm 0.6^{\dagger}$
AdaLNet	$81.9 \pm 1.9^{\dagger}$	$70.6 \pm 0.8^{\dagger}$	$77.8 \pm 0.7^{\dagger}$
DGI	82.5 ± 0.7	71.6 ± 0.7	78.4 ± 0.7
SGC	81.0 ± 0.0	71.9 ± 0.1	78.9 ± 0.0

Table 3. Test Micro F1 Score (%) averaged over 10 runs on Reddit. Performances of models are cited from their original papers. **OOM:** Out of memory.

Setting	Model	Test F1
Supervised	GaAN	96.4
	SAGE-mean	95.0
	SAGE-LSTM	95.4
	SAGE-GCN	93.0
	FastGCN	93.7
	GCN	**OOM**
Unsupervised	SAGE-mean	89.7
	SAGE-LSTM	90.7
	SAGE-GCN	90.8
	DGI	94.0
No Learning	Random-Init DGI	93.3
	SGC	94.9

Baselines. For citation networks, we compare against GCN (Kipf & Welling, 2017) GAT (Velickovic et al., 2018) FastGCN (Chen et al., 2018) LNet, AdaLNet (Liao et al., 2019) and DGI (Velikovi et al., 2019) using the publicly released implementations. Since GIN is not initially evaluated on citation networks, we implement GIN following Xu et al. (2019) and use hyperopt to tune weight decay and learning rate for 60 iterations. Moreover, we tune the hidden dimension by hand.

For Reddit, we compare SGC to the reported performance of GaAN (Zhang et al., 2018a), supervised and unsupervised variants of GraphSAGE (Hamilton et al., 2017), FastGCN, and DGI. Table 3 also highlights the setting of the feature extraction step for each method. We note that SGC involves no learning because the feature extraction step, $\mathbf{S}^K\mathbf{X}$, has no parameter. Both unsupervised and no-learning approaches train logistic regression models with labels afterward.

Performance. Based on results in Table 2 and Table 3, we conclude that SGC is very competitive. Table 2 shows the performance of SGC can match the performance of GCN and state-of-the-art graph networks on citation networks. In particular on Citeseer, SGC is about 1% better than GCN, and we reason this performance boost is caused by SGC having fewer parameters and therefore suffering less from overfitting. Remarkably, GIN performs slight worse because of overfitting. Also, both LNet and AdaLNet are unstable on citation networks. On Reddit, Table 3 shows that SGC outperforms the previous sampling-based GCN variants, SAGE-GCN and FastGCN by more than 1%.

Notably, Velikovi et al. (2019) report that the performance of a randomly initialized DGI encoder nearly matches that of a trained encoder; however, both models underperform SGC on Reddit. This result may suggest that the extra weights and nonlinearities in the DGI encoder are superfluous, if not outright detrimental.

Efficiency. In Figure 3, we plot the performance of the state-of-the-arts graph networks over their training time relative to that of SGC on the Pubmed and Reddit datasets. In particular, we precompute $\mathbf{S}^K\mathbf{X}$ and the training time of SGC takes into account this precomputation time. We measure the training time on a NVIDIA GTX 1080 Ti GPU and present the benchmark details in supplementary materials.

On large graphs (e.g. Reddit), GCN cannot be trained due to excessive memory requirements. Previous approaches tackle this limitation by either sampling to reduce neighborhood size (Chen et al., 2018; Hamilton et al., 2017) or limiting their model sizes (Velikovi et al., 2019). By applying a fixed filter and precomputing $\mathbf{S}^K\mathbf{X}$, SGC minimizes memory usage and only learns a single weight matrix during training. Since $\mathbf{S}$ is typically sparse and K is usually small, we can exploit fast sparse-dense matrix multiplication to compute $\mathbf{S}^K\mathbf{X}$. Figure 3 shows that SGC can be trained up to two orders of magnitude faster than fast sampling-based methods while having little or no drop in performance.

Table 4. Test Accuracy (%) on text classification datasets. The numbers are averaged over 10 runs.

Dataset	Model	Test Acc. ↑	Time (seconds) ↓
20NG	GCN	87.9 ± 0.2	1205.1 ± 144.5
	SGC	88.5 ± 0.1	19.06 ± 0.15
R8	GCN	97.0 ± 0.2	129.6 ± 9.9
	SGC	97.2 ± 0.1	1.90 ± 0.03
R52	GCN	93.8 ± 0.2	245.0 ± 13.0
	SGC	94.0 ± 0.2	3.01 ± 0.01
Ohsumed	GCN	68.2 ± 0.4	252.4 ± 14.7
	SGC	68.5 ± 0.3	3.02 ± 0.02
MR	GCN	76.3 ± 0.3	16.1 ± 0.4
	SGC	75.9 ± 0.3	4.00 ± 0.04

Table 5. Test accuracy (%) within 161 miles on semi-supervised user geolocation. The numbers are averaged over 5 runs.

Dataset	Model	Acc.@161↑	Time ↓
GEOTEXT	GCN+H	60.6 ± 0.2	153.0s
	SGC	61.1 ± 0.1	5.6s
TWITTER-US	GCN+H	61.9 ± 0.2	9h 54m
	SGC	62.5 ± 0.1	4h 33m
TWITTER-WORLD	GCN+H	53.6 ± 0.2	2d 05h 17m
	SGC	54.1 ± 0.2	22h 53m

5.2. Downstream Tasks

We extend our empirical evaluation to 5 downstream applications — text classification, semi-supervised user geolocation, relation extraction, zero-shot image classification, and graph classification — to study the applicability of SGC. We describe experimental setup in supplementary materials.

Text classification assigns labels to documents. Yao et al. (2019) use a 2-layer GCN to achieve state-of-the-art results by creating a corpus-level graph which treats both documents and words as nodes in a graph. Word-word edge weights are pointwise mutual information (PMI) and word-document edge weights are normalized TF-IDF scores. Table 4 shows that an SGC ($K = 2$) rivals their model on 5 benchmark datasets, while being up to $83.6\times$ faster.

Semi-supervised user geolocation locates the "home" position of users on social media given users' posts, connections among users, and a small number of labelled users. Rahimi et al. (2018) apply GCNs with highway connections on this task and achieve close to state-of-the-art results. Ta-

Table 6. Test Accuracy (%) on Relation Extraction. The numbers are averaged over 10 runs.

TACRED	Test Accuracy ↑
C-GCN (Zhang et al., 2018c)	66.4
C-GCN	66.4 ± 0.4
C-SGC	67.0 ± 0.4

Table 7. Top-1 accuracy (%) averaged over 10 runs in the 2-hop and 3-hop setting of the zero-shot image task on ImageNet. ADGPM (Kampffmeyer et al., 2018) and EXEM 1-nns (Changpinyo et al., 2018) use more powerful visual features.

Model	# Param. ↓	2-hop Acc. ↑	3-hop Acc. ↑
Unseen categories only:			
EXEM 1-nns	-	27.0	7.1
ADGPM	-	26.6	6.3
GCNZ	-	19.8	4.1
GCNZ (ours)	$9.5M$	20.9 ± 0.2	4.3 ± 0.0
MLP-SGCZ (ours)	$4.3M$	21.2 ± 0.2	4.4 ± 0.1
Unseen categories & seen categories:			
ADGPM	-	10.3	2.9
GCNZ	-	9.7	2.2
GCNZ (ours)	$9.5M$	10.0 ± 0.2	2.4 ± 0.0
MLP-SGCZ (ours)	$4.3M$	10.5 ± 0.1	2.5 ± 0.0

ble 5 shows that SGC outperforms GCN with highway connections on GEOTEXT (Eisenstein et al., 2010), TWITTER-US (Roller et al., 2012), and TWITTER-WORLD (Han et al., 2012) under Rahimi et al. (2018)'s framework, while saving 30+ hours on TWITTER-WORLD.

Relation extraction involves predicting the relation between subject and object in a sentence. Zhang et al. (2018c) propose C-GCN which uses an LSTM (Hochreiter & Schmidhuber, 1997) followed by a GCN and an MLP. We replace GCN with SGC ($K = 2$) and call the resulting model C-SGC. Table 6 shows that C-SGC sets new state-of-the-art on TACRED (Zhang et al., 2017).

Zero-shot image classification consists of learning an image classifier without access to any images or labels from the test categories. GCNZ (Wang et al., 2018) uses a GCN to map the category names — based on their relations in WordNet (Miller, 1995) — to image feature domain, and find the most similar category to a query image feature vector. Table 7 shows that replacing GCN with an MLP followed by SGC can improve performance while reducing the number of parameters by 55%. We find that an MLP feature extractor is necessary in order to map the pretrained GloVe vectors to the space of visual features extracted by a ResNet-50. Again, this downstream application demonstrates that learned graph convolution filters are superfluous; similar to Changpinyo et al. (2018)'s observation that GCNs may not be necessary.

Graph classification requires models to use graph structure to categorize graphs. Xu et al. (2019) theoretically show that GCNs are not sufficient to distinguish certain graph structures and show that their GIN is more expressive and achieves state-of-the-art results on various graph classification datasets. We replace the GCN in DCGCN (Zhang et al., 2018b) with an SGC and get 71.0% and 76.2% on NCI1 and COLLAB datasets (Yanardag & Vishwanathan, 2015) respectively, which is on par with an GCN counterpart, but far behind GIN. Similarly, on QM8 quantum chemistry dataset (Ramakrishnan et al., 2015), more advanced AdaLNet and LNet (Liao et al., 2019) get 0.01 MAE on QM8, outperforming SGC's 0.03 MAE by a large margin.

6. Conclusion

In order to better understand and explain the mechanisms of GCNs, we explore the simplest possible formulation of a graph convolutional model, SGC. The algorithm is almost trivial, a graph based pre-processing step followed by standard multi-class logistic regression. However, the performance of SGC rivals — if not surpasses — the performance of GCNs and state-of-the-art graph neural network models across a wide range of graph learning tasks. Moreover by precomputing the fixed feature extractor $\mathbf{S}^K$, training time is reduced to a record low. For example on the Reddit dataset, SGC can be trained up to two orders of magnitude faster than sampling-based GCN variants.

In addition to our empirical analysis, we analyze SGC from a convolution perspective and manifest this method as a low-pass-type filter on the spectral domain. Low-pass-type filters capture low-frequency signals, which corresponds with smoothing features across a graph in this setting. Our analysis also provides insight into the empirical boost of the "renormalization trick" and demonstrates how shrinking the spectral domain leads to a low-pass-type filter which underpins SGC.

Ultimately, the strong performance of SGC sheds light onto GCNs. It is likely that the expressive power of GCNs originates primarily from the repeated graph propagation (which SGC preserves) rather than the nonlinear feature extraction (which it doesn't.)

Given its empirical performance, efficiency, and interpretability, we argue that the SGC should be highly beneficial to the community in at least three ways: (1) as a first model to try, especially for node classification tasks; (2) as a simple baseline for comparison with future graph learning models; (3) as a starting point for future research in graph learning — returning to the historic machine learning practice to develop complex from simple models.

Acknowledgement

This research is supported in part by grants from the National Science Foundation (III-1618134, III-1526012, IIS1149882, IIS-1724282, and TRIPODS-1740822), the Office of Naval Research DOD (N00014-17-1-2175), Bill and Melinda Gates Foundation, and Facebook Research. We are thankful for generous support by SAP America Inc. Amauri Holanda de Souza Jr. thanks CNPq (Brazilian Council for Scientific and Technological Development) for the financial support. We appreciate the discussion with Xiang Fu, Shengyuan Hu, Shangdi Yu, Wei-Lun Chao and Geoff Pleiss as well as the figure design support from Boyi Li.

References

Abu-El-Haija, S., Kapoor, A., Perozzi, B., and Lee, J. N-GCN: Multi-scale graph convolution for semi-supervised node classification. *arXiv preprint arXiv:1802.08888*, 2018.

Atwood, J. and Towsley, D. Diffusion-convolutional neural networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 1993–2001. Curran Associates, Inc., 2016.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Belkin, M. and Niyogi, P. Semi-supervised learning on riemannian manifolds. *Machine Learning*, 56(1-3):209–239, 2004.

Belkin, M., Niyogi, P., and Sindhwani, V. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. *Computational Science & Discovery*, 8(1), 2015.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of 19th International Conference on Computational Statistics*, pp. 177–186. Springer, 2010.

Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR'2014)*, 2014.

Cai, C. and Wang, Y. A simple yet effective baseline for non-attribute graph classification, 2018.

Changpinyo, S., Chao, W.-L., Gong, B., and Sha, F. Classifier and exemplar synthesis for zero-shot learning. *arXiv preprint arXiv:1812.06423*, 2018.

Chen, J., Ma, T., and Xiao, C. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *International Conference on Learning Representations (ICLR'2018)*, 2018.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3844–3852. Curran Associates, Inc., 2016.

Duran, A. G. and Niepert, M. Learning graph representations with embedding propagation. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5119–5130. Curran Associates, Inc., 2017.

Eisenstein, J., O'Connor, B., Smith, N. A., and Xing, E. P. A latent variable model for geographic lexical variation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 1277–1287. Association for Computational Linguistics, 2010.

Eliav, B. and Cohen, E. Bootstrapped graph diffusions: Exposing the power of nonlinearity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2 (1):10:1–10:19, 2018.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 1024–1034. Curran Associates, Inc., 2017.

Han, B., Cook, P., and Baldwin, T. Geolocation prediction in social media data by finding location indicative words. In *Proceedings of the 24th International Conference on Computational Linguistics*, pp. 1045–1062, 2012.

Hariharan, B., Arbeláez, P., Girshick, R., and Malik, J. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 447–456, 2015.

Harris, C. and Stephens, M. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pp. 147–151, 1988.

Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*, 9:1735–1780, 1997.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pp. 646–661. Springer, 2016.

Kampffmeyer, M., Chen, Y., Liang, X., Wang, H., Zhang, Y., and Xing, E. P. Rethinking knowledge graph propagation for zero-shot learning. *arXiv preprint arXiv:1805.11724*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR'2015)*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR'2017)*, 2017.

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

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Lee, J. B., Rossi, R. A., Kim, S., Ahmed, N. K., and Koh, E. Attention models in graphs: A survey. *arXiv e-prints*, 2018.

Li, Q., Han, Z., and Wu, X. Deeper insights into graph convolutional networks for semi-supervised learning. *CoRR*, abs/1801.07606, 2018.

Li, Q., Wu, X.-M., Liu, H., Zhang, X., and Guan, Z. Label efficient semi-supervised learning via graph filtering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Liao, R., Zhao, Z., Urtasun, R., and Zemel, R. Lanczosnet: Multi-scale deep graph convolutional networks. In *International Conference on Learning Representations (ICLR'2019)*, 2019.

Liu, D. C. and Nocedal, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989.

Miller, G. A. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 5425–5434, 2017.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'14, pp. 701–710. ACM, 2014.

Rahimi, S., Cohn, T., and Baldwin, T. Semi-supervised user geolocation via graph convolutional networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2009–2019. Association for Computational Linguistics, 2018.

Ramakrishnan, R., Hartmann, M., Tapavicza, E., and von Lilienfeld, O. A. Electronic spectra from TDDFT and machine learning in chemical space. *The Journal of chemical physics*, 143(8):084111, 2015.

Roller, S., Speriosu, M., Rallapalli, S., Wing, B., and Baldridge, J. Supervised text-based geolocation using language models on an adaptive grid. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 1500–1510. Association for Computational Linguistics, 2012.

Rosenblatt, F. The Perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Rosenblatt, F. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc, 1961.

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

Sobel, I. and Feldman, G. A 3x3 isotropic gradient operator for image processing. *A talk at the Stanford Artificial Project*, pp. 271–272, 1968.

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

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. In *International Conference on Learning Representations (ICLR'2018)*, 2018.

Velikovi, P., Fedus, W., Hamilton, W. L., Li, P., Bengio, Y., and Hjelm, R. D. Deep Graph InfoMax. In *International Conference on Learning Representations (ICLR'2019)*, 2019.

Waibel, A., Hanazawa, T., and Hinton, G. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech. and Signal Processing*, 37(3), 1989.

Wang, X., Ye, Y., and Gupta, A. Zero-shot recognition via semantic embeddings and knowledge graphs. In *Computer Vision and Pattern Recognition (CVPR)*, pp. 6857–6866. IEEE Computer Society, 2018.

Weston, J., Ratle, F., and Collobert, R. Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 1168–1175, 2008.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

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

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374. ACM, 2015.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 40–48, 2016.

Yao, L., Mao, C., and Luo, Y. Graph convolutional networks for text classification. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*, 2019.

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 *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI'2018)*, pp. 339–349. AUAI Press, 2018a.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. 2018b.

Zhang, Y., Zhong, V., Chen, D., Angeli, G., and Manning, C. D. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 35–45. Association for Computational Linguistics, 2017.

Zhang, Y., Qi, P., and Manning, C. D. Graph convolution over pruned dependency trees improves relation extraction. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018c.

Zhou, D., Bousquet, O., Thomas, N. L., Weston, J., and Schölkopf, B. Learning with local and global consistency. In Thrun, S., Saul, L. K., and Schölkopf, B. (eds.), *Advances in Neural Information Processing Systems 16*, pp. 321–328. MIT Press, 2004.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. Graph Neural Networks: A Review of Methods and Applications. *arXiv e-prints*, 2018.

Zhu, X., Ghahramani, Z., and Lafferty, J. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pp. 912–919. AAAI Press, 2003.

Simplifying Graph Convolutional Networks (Supplementary Material)

A. The spectrum of $\tilde{\mathbf{\Delta}}_{\text{sym}}$

The normalized Laplacian defined on graphs with self-loops, $\tilde{\mathbf{\Delta}}_{\text{sym}}$, consists of an instance of generalized graph Laplacians and hold the interpretation as a difference operator, i.e. for any signal $\mathbf{x} \in \mathbb{R}^n$ it satisfies

$$(\tilde{\mathbf{\Delta}}_{\text{sym}}\mathbf{x})_i = \sum_j \frac{\tilde{a}_{ij}}{\sqrt{d_i+\gamma}}\left(\frac{x_i}{\sqrt{d_i+\gamma}} - \frac{x_j}{\sqrt{d_j+\gamma}}\right).$$

Here, we prove several properties regarding its spectrum.

Lemma 1. *(Non-negativity of $\tilde{\mathbf{\Delta}}_{sym}$) The augmented normalized Laplacian matrix is symmetric positive semi-definite.*

Proof. The quadratic form associated with $\tilde{\mathbf{\Delta}}_{\text{sym}}$ is

$$\begin{aligned}
\mathbf{x}^\top \tilde{\mathbf{\Delta}}_{\text{sym}}\mathbf{x} &= \sum_i x_i^2 - \sum_i \sum_j \frac{\tilde{a}_{ij}x_i x_j}{\sqrt{(d_i+\gamma)(d_j+\gamma)}} \\
&= \frac{1}{2}\left(\sum_i x_i^2 + \sum_j x_j^2 - \sum_i \sum_j \frac{2\tilde{a}_{ij}x_i x_j}{\sqrt{(d_i+\gamma)(d_j+\gamma)}}\right) \\
&= \frac{1}{2}\left(\sum_i \sum_j \frac{\tilde{a}_{ij}x_i^2}{d_i+\gamma} + \sum_j \sum_i \frac{\tilde{a}_{ij}x_j^2}{d_j+\gamma}\right. \\
&\qquad \left. - \sum_i \sum_j \frac{2\tilde{a}_{ij}x_i x_j}{\sqrt{(d_i+\gamma)(d_j+\gamma)}}\right) \\
&= \frac{1}{2}\sum_i \sum_j \tilde{a}_{ij}\left(\frac{x_i}{\sqrt{d_i+\gamma}} - \frac{x_j}{\sqrt{d_j+\gamma}}\right)^2 \geq 0 \qquad (13)
\end{aligned}$$

□

Lemma 2. 0 *is an eigenvalue of both $\mathbf{\Delta}_{sym}$ and $\tilde{\mathbf{\Delta}}_{sym}$.*

Proof. First, note that $\mathbf{v} = [1, \ldots, 1]^\top$ is an eigenvector of $\mathbf{\Delta}$ associated with eigenvalue 0, i.e., $\mathbf{\Delta}\mathbf{v} = (\mathbf{D} - \mathbf{A})\mathbf{v} = \mathbf{0}$.

Also, we have that $\tilde{\mathbf{\Delta}}_{\text{sym}} = \tilde{\mathbf{D}}^{-1/2}(\tilde{\mathbf{D}} - \tilde{\mathbf{A}})\tilde{\mathbf{D}}^{-1/2} = \tilde{\mathbf{D}}^{-1/2}\mathbf{\Delta}\tilde{\mathbf{D}}^{-1/2}$. Denote $\mathbf{v}_1 = \tilde{\mathbf{D}}^{1/2}\mathbf{v}$, then

$$\tilde{\mathbf{\Delta}}_{\text{sym}}\mathbf{v}_1 = \tilde{\mathbf{D}}^{-1/2}\mathbf{\Delta}\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{D}}^{1/2}\mathbf{v} = \tilde{\mathbf{D}}^{-1/2}\mathbf{\Delta}\mathbf{v} = \mathbf{0}.$$

Therefore, $\mathbf{v}_1 = \tilde{\mathbf{D}}^{1/2}\mathbf{v}$ is an eigenvector of $\tilde{\mathbf{\Delta}}_{\text{sym}}$ associated with eigenvalue 0, which is then the smallest eigenvalue from the non-negativity of $\tilde{\mathbf{\Delta}}_{\text{sym}}$. Likewise, 0 can be proved to be the smallest eigenvalues of $\mathbf{\Delta}_{\text{sym}}$. □

Lemma 3. *Let $\beta_1 \leq \beta_2 \leq \cdots \leq \beta_n$ denote eigenvalues of $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ and $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_n$ be the eigenvalues of $\tilde{\mathbf{D}}^{-1/2}\mathbf{A}\tilde{\mathbf{D}}^{-1/2}$. Then,*

$$\alpha_1 \geq \frac{\max_i d_i}{\gamma + \max_i d_i}\beta_1, \qquad \alpha_n \leq \frac{\min_i d_i}{\gamma + \min_i d_i}. \qquad (14)$$

Proof. We have shown that 0 is an eigenvalue of $\mathbf{\Delta}_{\text{sym}}$. Since $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{\Delta}_{\text{sym}}$, then 1 is an eigenvalue of $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. More specifically, $\beta_n = 1$. In addition, by combining the fact that $\text{Tr}(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}) = 0 = \sum_i \beta_i$ with $\beta_n = 1$, we conclude that $\beta_1 < 0$.

By choosing $\mathbf{x}$ such that $\|\mathbf{x}\| = 1$ and $\mathbf{y} = \mathbf{D}^{1/2}\tilde{\mathbf{D}}^{-1/2}\mathbf{x}$, we have that $\|\mathbf{y}\|^2 = \sum_i \frac{d_i}{d_i+\gamma}x_i^2$ and $\frac{\min_i d_i}{\gamma+\min_i d_i} \leq \|\mathbf{y}\|^2 \leq \frac{\max_i d_i}{\gamma+\max_i d_i}$. Hence, we use the Rayleigh quotient to provide a lower bound to α_1:

$$\begin{aligned}
\alpha_1 &= \min_{\|\mathbf{x}\|=1}\left(\mathbf{x}^\top\tilde{\mathbf{D}}^{-1/2}\mathbf{A}\tilde{\mathbf{D}}^{-1/2}\mathbf{x}\right) \\
&= \min_{\|\mathbf{x}\|=1}\left(\mathbf{y}^\top\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{y}\right) \text{ (by replacing variable)} \\
&= \min_{\|\mathbf{x}\|=1}\left(\frac{\mathbf{y}^\top\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{y}}{\|\mathbf{y}\|^2}\|\mathbf{y}\|^2\right) \\
&\geq \min_{\|\mathbf{x}\|=1}\left(\frac{\mathbf{y}^\top\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{y}}{\|\mathbf{y}\|^2}\right)\max_{\|\mathbf{x}\|=1}\left(\|\mathbf{y}\|^2\right) \\
&\quad (\because \min(AB) \geq \min(A)\max(B) \text{ if } \min(A) < 0, \forall B > 0, \\
&\quad \text{and} \quad \min_{\|\mathbf{x}\|=1}\left(\frac{\mathbf{y}^\top\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{y}}{\|\mathbf{y}\|^2}\right) = \beta_1 < 0) \\
&= \beta_1 \max_{\|\mathbf{x}\|=1}\|\mathbf{y}\|^2 \\
&\geq \frac{\max_i d_i}{\gamma + \max_i d_i}\beta_1.
\end{aligned}$$

One may employ similar steps to prove the second inequality in Equation 14.

□

Proof of Theorem 1. Note that $\tilde{\mathbf{\Delta}}_{\text{sym}} = \mathbf{I} - \gamma\tilde{\mathbf{D}}^{-1} - \tilde{\mathbf{D}}^{-1/2}\mathbf{A}\tilde{\mathbf{D}}^{-1/2}$. Using the results in Lemma 3, we show

that the largest eigenvalue $\tilde{\lambda}_n$ of $\tilde{\mathbf{\Delta}}_{\text{sym}}$ is

$$\begin{aligned}
\tilde{\lambda}_n &= \max_{\|\mathbf{x}\|=1} \mathbf{x}^\top (\mathbf{I} - \gamma \tilde{\mathbf{D}}^{-1} - \tilde{\mathbf{D}}^{-1/2} \mathbf{A} \tilde{\mathbf{D}}^{-1/2}) \mathbf{x} \\
&\leq 1 - \min_{\|\mathbf{x}\|=1} \gamma \mathbf{x}^\top \tilde{\mathbf{D}}^{-1} \mathbf{x} - \min_{\|\mathbf{x}\|=1} \mathbf{x}^\top \tilde{\mathbf{D}}^{-1/2} \mathbf{A} \tilde{\mathbf{D}}^{-1/2} \mathbf{x} \\
&= 1 - \frac{\gamma}{\gamma + \max_i d_i} - \alpha_1 \\
&\leq 1 - \frac{\gamma}{\gamma + \max_i d_i} - \frac{\max_i d_i}{\gamma + \max_i d_i} \beta_1 \\
&< 1 - \frac{\max_i d_i}{\gamma + \max_i d_i} \beta_1 \quad (\gamma > 0 \text{ and } \beta_1 < 0) \\
&< 1 - \beta_1 = \lambda_n \qquad (15)
\end{aligned}$$

□

B. Experiment Details

Node Classification. We empirically find that on Reddit dataset for SGC, it is crucial to normalize the features into zero mean and univariate.

Training Time Benchmarking. We hereby describe the experiment setup of Figure 3. Chen et al. (2018) benchmark the training time of FastGCN on CPU, and as a result, it is difficult to compare numerical values across reports. Moreover, we found the performance of FastGCN improved with a smaller early stopping window (10 epochs); therefore, we could decrease the model's training time. We provide the data underpinning Figure 3 in Table 8 and Table 9.

Table 8. Training time (seconds) of graph neural networks on Citation Networks. Numbers are averaged over 10 runs.

Models	Cora	Citeseer	Pubmed
GCN	0.49	0.59	8.31
GAT	63.10	118.10	121.74
FastGCN	2.47	3.96	1.77
GIN	2.09	4.47	26.15
LNet	15.02	49.16	266.47
AdaLNet	10.15	31.80	222.21
DGI	21.24	21.06	76.20
SGC	0.13	0.14	0.29

Table 9. Training time (seconds) on Reddit dataset.

Model	Time(s) ↓
SAGE-mean	78.54
SAGE-LSTM	486.53
SAGE-GCN	86.86
FastGCN	270.45
SGC	2.70

Text Classification. Yao et al. (2019) use one-hot features for the word and document nodes. In training SGC, we normalize the features to be between 0 and 1 **after propagation** and train with L-BFGS for 3 steps. We tune the only hyperparameter, weight decay, using hyperopt(Bergstra et al., 2015) for 60 iterations. Note that we cannot apply this feature normalization for TextGCN because the propagation cannot be precomputed.

Semi-supervised User Geolocation. We replace the 4-layer, highway-connection GCN with a 3rd degree propagation matrix ($K = 3$) SGC and use the same set of hyperparameters as Rahimi et al. (2018). All experiments on the GEOTEXT dataset are conducted on a single Nvidia GTX-1080Ti GPU while the ones on the TWITTER-NA and TWITTER-WORLD datasets are excuded with 10 cores of the Intel(R) Xeon(R) Silver 4114 CPU (2.20GHz). Instead of collapsing all linear transformations, we keep two of them which we find performing slightly better possibly due to . Despite of this subtle variation, the model is still linear.

Relation Extraction. We replace the 2-layer GCN with a 2nd degree propagation matrix ($K = 2$) SGC and remove the intermediate dropout. We keep other hyperparameters unchanged, including learning rate and regularization. Similar to Zhang et al. (2018c), we report the best validation accuracy with early stopping.

Zero-shot Image Classification. We replace the 6-layer GCN (hidden size: 2048, 2048, 1024, 1024, 512, 2048) baseline with an 6-layer MLP (hidden size: 512, 512, 512, 1024, 1024, 2048) followed by a SGC with $K = 6$. Following (Wang et al., 2018), we only apply dropout to the output of SGC. Due to the slow evaluation of this task, we do not tune the dropout rate or other hyperparameters. Rather, we follow the GCNZ code and use learning rate of 0.001, weight decay of 0.0005, and dropout rate of 0.5. We also train the models with ADAM (Kingma & Ba, 2015) for 300 epochs.

C. Additional Experiments

Random Splits for Citation Networks. Possibly due to their limited size, the citation networks are known to be unstable. Accordingly, we conduct an additional 10 experiments on random splits of the training set while maintaining the same validation and test sets.

Propagation choice. We conduct an ablation study with different choices of propagation matrix, namely:

Normalized Adjacency: $\mathbf{S}_{\text{adj}} = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$

Random Walk Adjacency $\mathbf{S}_{\text{rw}} = \mathbf{D}^{-1} \mathbf{A}$

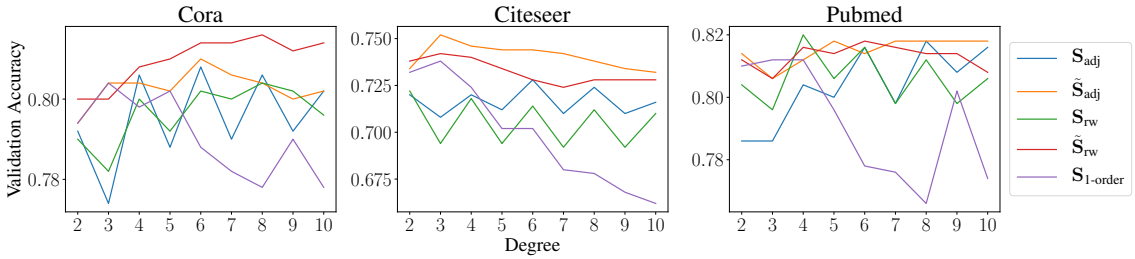

Figure 4. Validation accuracy with SGC using different propagation matrices.

Table 10. Test accuracy (%) on citation networks (random splits). †We remove the outliers (accuracy $< 0.7/0.65/0.75$) when calculating their statistics due to high variance.

	Cora	Citeseer	Pubmed
Ours:			
GCN	80.53 ± 1.40	70.67 ± 2.90	77.09 ± 2.95
GIN	76.94 ± 1.24	66.56 ± 2.27	74.46 ± 2.19
LNet	$74.23 \pm 4.50^{\dagger}$	$67.26 \pm 0.81^{\dagger}$	$77.20 \pm 2.03^{\dagger}$
AdaLNet	$72.68 \pm 1.45^{\dagger}$	$71.04 \pm 0.95^{\dagger}$	$77.53 \pm 1.76^{\dagger}$
GAT	82.29 ± 1.16	72.6 ± 0.58	78.79 ± 1.41
SGC	80.62 ± 1.21	71.40 ± 3.92	77.02 ± 1.62

Aug. Normalized Adjacency $\tilde{\mathbf{S}}_{\text{adj}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$

Aug. Random Walk $\tilde{\mathbf{S}}_{\text{rw}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$

First-Order Cheby $\mathbf{S}_{\text{1-order}} = (\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})$

We investigate the effect of propagation steps $K \in \{2..10\}$ on validation set accuracy. We use hyperopt to tune L2-regularization and leave all other hyperparameters unchanged. Figure 4 depicts the validation results achieved by varying the degree of different propagation matrices.

We see that augmented propagation matrices (i.e. those with self-loops) attain higher accuracy and more stable performance across various propagation depths. Specifically, the accuracy of $\mathbf{S}_{\text{1-order}}$ tends to deteriorate as the power K increases, and this results suggests using large filter coefficients on low frequencies degrades SGC performance on semi-supervised tasks.

Another pattern is that odd powers of K cause a significant performance drop for the normalized adjacency and random walk propagation matrices. This demonstrates how odd powers of the un-augmented propagation matrix use negative filter coefficients on high frequency information. Adding self-loops to the propagation matrix shrinks the spectrum such that the largest eigenvalues decrease from ≈ 2 to ≈ 1.5 on the citation network datasets. By effectively shrinking the spectrum, the effect of negative filter coefficients on high frequencies is minimized, and as a result, using odd-powers of K does not degrade the performance of augmented propagation matrices. For non-augmented propagation matrices — where the largest eigenvalue is approximately 2 — negative coefficients significantly distort the signal, which leads to decreased accuracy. Therefore, adding self-loops constructs a better domain in which fixed filters can operate.

# Training Samples	SGC	GCN
1	33.16	32.94
5	63.74	60.68
10	72.04	71.46
20	80.30	80.16
40	85.56	85.38
80	90.08	90.44

Table 11. Validation Accuracy (%) when SGC and GCN are trained with different amounts of data on Cora. The validation accuracy is averaged over 10 random training splits such that each class has the same number of training examples.

Data amount. We also investigated the effect of training dataset size on accuracy. As demonstrated in Table 11, SGC continues to perform similarly to GCN as the training dataset size is reduced, and even outperforms GCN when there are fewer than 5 training samples. We reason this study demonstrates SGC has at least the same modeling capacity as GCN.