

# Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning

Qimai Li,<sup>1</sup> Zhichao Han,<sup>1,2</sup> Xiao-Ming Wu<sup>1\*</sup>

<sup>1</sup>The Hong Kong Polytechnic University

<sup>2</sup>ETH Zurich

csqqli@comp.polyu.edu.hk, zhhan@student.ethz.ch, xiao-ming.wu@polyu.edu.hk

## Abstract

Many interesting problems in machine learning are being revisited with new deep learning tools. For graph-based semi-supervised learning, a recent important development is graph convolutional networks (GCNs), which nicely integrate local vertex features and graph topology in the convolutional layers. Although the GCN model compares favorably with other state-of-the-art methods, its mechanisms are not clear and it still requires considerable amount of labeled data for validation and model selection.

In this paper, we develop deeper insights into the GCN model and address its fundamental limits. First, we show that the graph convolution of the GCN model is actually a special form of Laplacian smoothing, which is the key reason why GCNs work, but it also brings potential concerns of over-smoothing with many convolutional layers. Second, to overcome the limits of the GCN model with shallow architectures, we propose both co-training and self-training approaches to train GCNs. Our approaches significantly improve GCNs in learning with very few labels, and exempt them from requiring additional labels for validation. Extensive experiments on benchmarks have verified our theory and proposals.

## 1 Introduction

The breakthroughs in deep learning have led to a paradigm shift in artificial intelligence and machine learning. On the one hand, numerous old problems have been revisited with deep neural networks and huge progress has been made in many tasks previously seemed out of reach, such as machine translation and computer vision. On the other hand, new techniques such as geometric deep learning (Bronstein et al. 2017) are being developed to generalize deep neural models to new or non-traditional domains.

It is well known that training a deep neural model typically requires a large amount of labeled data, which cannot be satisfied in many scenarios due to the high cost of labeling training data. To reduce the amount of data needed for training, a recent surge of research interest has focused on few-shot learning (Lake, Salakhutdinov, and Tenenbaum 2015; Rezende et al. 2016) – to learn a classification model with very few examples from each class. Closely related to few-shot learning is semi-supervised learning, where a large

amount of *unlabeled* data can be utilized to train with typically a small amount of labeled data.

Many researches have shown that leveraging unlabeled data in training can improve learning accuracy significantly if used properly (Zhu and Goldberg 2009). The key issue is to maximize the effective utilization of structural and feature information of unlabeled data. Due to the powerful feature extraction capability and recent success of deep neural networks, there have been some successful attempts to revisit semi-supervised learning with neural-network-based models, including ladder network (Rasmus et al. 2015), semi-supervised embedding (Weston et al. 2008), planetoid (Yang, Cohen, and Salakhutdinov 2016), and graph convolutional networks (Kipf and Welling 2017).

The recently developed graph convolutional neural networks (GCNNs) (Defferrard, Bresson, and Vandergheynst 2016) is a successful attempt of generalizing the powerful convolutional neural networks (CNNs) in dealing with Euclidean data to modeling graph-structured data. In their pilot work (Kipf and Welling 2017), Kipf and Welling proposed a simplified type of GCNNs, called graph convolutional networks (GCNs), and applied it to semi-supervised classification. The GCN model naturally integrates the connectivity patterns and feature attributes of graph-structured data, and outperforms many state-of-the-art methods significantly on some benchmarks. Nevertheless, it suffers from similar problems faced by other neural-network-based models. The working mechanisms of the GCN model for semi-supervised learning are not clear, and the training of GCNs still requires considerable amount of labeled data for parameter tuning and model selection, which defeats the purpose for semi-supervised learning.

In this paper, we demystify the GCN model for semi-supervised learning. In particular, we show that the graph convolution of the GCN model is simply a special form of Laplacian smoothing, which mixes the features of a vertex and its nearby neighbors. The smoothing operation makes the features of vertices in the same cluster similar, thus greatly easing the classification task, which is the key reason why GCNs work so well. However, it also brings potential concerns of over-smoothing. If a GCN is deep with many convolutional layers, the output features may be over-smoothed and vertices from different clusters may become indistinguishable. The mixing happens quickly on small

\*Corresponding author.

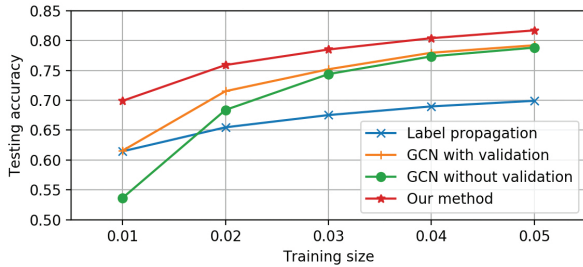


Figure 1: Performance comparison of GCNs, label propagation, and our method for semi-supervised classification on the Cora citation network.

datasets with only a few convolutional layers, as illustrated by Fig. 2. Also, adding more layers to a GCN will make it much more difficult to train.

However, a shallow GCN model such as the two-layer GCN used in (Kipf and Welling 2017) has its own limits. Besides that it requires many additional labels for validation, it also suffers from the localized nature of the convolutional filter. When only few labels are given, a shallow GCN cannot effectively propagate the labels to the entire data graph. As illustrated in Fig. 1, the performance of GCNs drops quickly as the training size shrinks, even for the one with 500 additional labels for validation.

To overcome the limits and realize the full potentials of the GCN model, we propose a co-training approach and a self-training approach to train GCNs. By co-training a GCN with a random walk model, the latter could complement the former in exploring global graph topology. By self-training a GCN, we can exploit its feature extraction capability to overcome its localized nature. Combining both the co-training and self-training approaches can substantially improve the GCN model for semi-supervised learning with very few labels, and exempt it from requiring additional labeled data for validation. As illustrated in Fig. 1, our method outperforms GCNs by a large margin.

In a nutshell, the key innovations of this paper are: 1) providing new insights and analysis of the GCN model for semi-supervised learning; 2) proposing solutions to improve the GCN model for semi-supervised learning. The rest of the paper is organized as follows. Section 2 introduces the preliminaries and related works. In Section 3, we analyze the mechanisms and fundamental limits of the GCN model for semi-supervised learning. In Section 4, we propose our methods to improve the GCN model. In Section 5, we conduct experiments to verify our analysis and proposals. Finally, Section 6 concludes the paper.

## 2 Preliminaries and Related Works

First, let us define some notations used throughout this paper. A graph is represented by  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V}$  is the vertex set with  $|\mathcal{V}| = n$  and  $\mathcal{E}$  is the edge set. In this paper, we consider undirected graphs. Denote by  $A = [a_{ij}] \in \mathbb{R}^{n \times n}$  the adjacency matrix which is nonnegative. Denote by  $D = \text{diag}(d_1, d_2, \dots, d_n)$  the degree matrix of  $A$  where

$d_i = \sum_j a_{ij}$  is the degree of vertex  $i$ . The graph Laplacian (Chung 1997) is defined as  $L := D - A$ , and the two versions of normalized graph Laplacians are defined as  $L_{\text{sym}} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$  and  $L_{\text{rw}} := D^{-1} L$  respectively.

## Graph-Based Semi-Supervised Learning

The problem we consider in this paper is semi-supervised classification on graphs. Given a graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$ , where  $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T \in \mathbb{R}^{n \times c}$  is the feature matrix, and  $\mathbf{x}_i \in \mathbb{R}^c$  is the  $c$ -dimensional feature vector of vertex  $i$ . Suppose that the labels of a set of vertices  $\mathcal{V}_l$  are given, the goal is to predict the labels of the remaining vertices  $\mathcal{V}_u$ .

Graph-based semi-supervised learning has been a popular research area in the past two decades. By exploiting the graph or manifold structure of data, it is possible to learn with very few labels. Many graph-based semi-supervised learning methods make the cluster assumption (Chapelle and Zien 2005), which assumes that nearby vertices on a graph tend to share the same label. Researches along this line include min-cuts (Blum and Chawla 2001) and randomized min-cuts (Blum et al. 2004), spectral graph transducer (Joachims 2003), label propagation (Zhu, Ghahramani, and Lafferty 2003) and its variants (Zhou et al. 2004; Bengio, Delalleau, and Le Roux 2006), modified adsorption (Talukdar and Crammer 2009), and iterative classification algorithm (Sen et al. 2008).

But the graph only represents the structural information of data. In many applications, data instances come with feature vectors containing information not present in the graph. For example, in a citation network, the citation links between documents describe their citation relations, while the documents are represented as bag-of-words vectors which describe their contents. Many semi-supervised learning methods seek to jointly model the graph structure and feature attributes of data. A common idea is to regularize a supervised learner with some regularizer. For example, manifold regularization (LapSVM) (Belkin, Niyogi, and Sindhwani 2006) regularizes a support vector machine with a Laplacian regularizer. Deep semi-supervised embedding (Weston et al. 2008) regularizes a deep neural network with an embedding-based regularizer. Planetoid (Yang, Cohen, and Salakhutdinov 2016) also regularizes a neural network by jointly predicting the class label and the context of an instance.

## Graph Convolutional Networks

Graph convolutional neural networks (GCNNs) generalize traditional convolutional neural networks to the graph domain. There are mainly two types of GCNNs (Bronstein et al. 2017): spatial GCNNs and spectral GCNNs. Spatial GCNNs view the convolution as ‘‘patch operator’’ which constructs a new feature vector for each vertex using its neighborhood information. Spectral GCNNs define the convolution by decomposing a graph signal  $\mathbf{s} \in \mathbb{R}^n$  (a scalar for each vertex) on the spectral domain and then applying a spectral filter  $g_\theta$  (a function of eigenvalues of  $L_{\text{sym}}$ ) on the spectral components (Bruna et al. 2014; Sandryhaila and Moura 2013; Shuman et al. 2013). However this model requires explicitly computing the Laplacian eigenvectors,

which is impractical for real large graphs. A way to circumvent this problem is by approximating the spectral filter  $g_\theta$  with Chebyshev polynomials up to  $K^{th}$  order (Hammond, Vandergheynst, and Gribonval 2011). In (Defferrard, Breuss, and Vandergheynst 2016), Defferrard et al. applied this to build a  $K$ -localized ChebNet, where the convolution is defined as:

$$g_\theta \star \mathbf{s} \approx \sum_{k=0}^K \theta'_k T_k(L_{\text{sym}}) \mathbf{s}, \quad (1)$$

where  $\mathbf{s} \in R^n$  is the signal on the graph,  $g_\theta$  is the spectral filter,  $\star$  denotes the convolution operator,  $T_k$  is the Chebyshev polynomials, and  $\theta' \in R^K$  is a vector of Chebyshev coefficients. By the approximation, the ChebNet is actually spectrum-free.

In (Kipf and Welling 2017), Kipf and Welling simplified this model by limiting  $K = 1$  and approximating the largest eigenvalue  $\lambda_{max}$  of  $L_{\text{sym}}$  by 2. In this way, the convolution becomes

$$g_\theta \star \mathbf{s} = \theta \left( I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) \mathbf{s}, \quad (2)$$

where  $\theta$  is the only Chebyshev coefficient left. They further applied a normalization trick to the convolution matrix:

$$I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}, \quad (3)$$

where  $\tilde{A} = A + I$  and  $\tilde{D} = \sum_j \tilde{A}_{ij}$ .

Generalizing the above definition of convolution to a graph signal with  $c$  input channels, i.e.,  $X \in R^{n \times c}$  (each vertex is associated with a  $c$ -dimensional feature vector), and using  $f$  spectral filters, the propagation rule of this simplified model is:

$$H^{(l+1)} = \sigma \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} \Theta^{(l)} \right), \quad (4)$$

where  $H^{(l)}$  is the matrix of activations in the  $l$ -th layer, and  $H^{(0)} = X$ ,  $\Theta^{(l)} \in R^{c \times f}$  is the trainable weight matrix in layer  $l$ ,  $\sigma$  is the activation function, e.g.,  $ReLU(\cdot) = \max(0, \cdot)$ .

This simplified model is called graph convolutional networks (GCNs), which is the focus of this paper.

### Semi-Supervised Classification with GCNs

In (Kipf and Welling 2017), the GCN model was applied for semi-supervised classification in a neat way. The model used is a two-layer GCN which applies a *softmax* classifier on the output features:

$$Z = \text{softmax} \left( \hat{A} ReLU \left( \hat{A} X \Theta^{(0)} \right) \Theta^{(1)} \right), \quad (5)$$

where  $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ ,  $\text{softmax}(x_i) = \frac{1}{Z} \exp(x_i)$  with  $Z = \sum_i \exp(x_i)$ . The loss function is defined as the cross-entropy error over all labeled examples:

$$\mathcal{L} := - \sum_{i \in \mathcal{V}_l} \sum_{f=1}^F Y_{if} \ln Z_{if}, \quad (6)$$

where  $\mathcal{V}_l$  is the set of indices of labeled vertices and  $F$  is the dimension of the output features, which is equal to the number of classes.  $Y \in R^{|\mathcal{V}_l| \times F}$  is a label indicator matrix. The

Table 1: GCNs vs. Fully-connected networks

One-layer FCN	Two-layer FCN	One-layer GCN	Two-layer GCN
0.530860	0.559260	<b>0.707940</b>	<b>0.798361</b>

weight parameters  $\Theta^{(0)}$  and  $\Theta^{(1)}$  can be trained via gradient descent.

The GCN model naturally combines graph structures and vertex features in the convolution, where the features of unlabeled vertices are mixed with those of nearby labeled vertices, and propagated over the graph through multiple layers. It was reported in (Kipf and Welling 2017) that GCNs outperformed many state-of-the-art methods significantly on some benchmarks such as citation networks.

## 3 Analysis

Despite its promising performance, the mechanisms of the GCN model for semi-supervised learning have not been made clear. In this section, we take a closer look at the GCN model, analyze why it works, and point out its limitations.

### Why GCNs Work

To understand the reasons why GCNs work so well, we compare them with the simplest fully-connected networks (FCNs), where the layer-wise propagation rule is

$$H^{(l+1)} = \sigma \left( H^{(l)} \Theta^{(l)} \right). \quad (7)$$

Clearly the only difference between a GCN and a FCN is the graph convolution matrix  $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$  (Eq. (5)) applied on the left of the feature matrix  $X$ . To see the impact of the graph convolution, we tested the performances of GCNs and FCNs for semi-supervised classification on the Cora citation network with 20 labels in each class. The results can be seen in Table 1. Surprisingly, even a one-layer GCN outperformed a one-layer FCN by a very large margin.

**Laplacian Smoothing.** Let us first consider a one-layer GCN. It actually contains two steps. 1) Generating a new feature matrix  $Y$  from  $X$  by applying the graph convolution:

$$Y = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X. \quad (8)$$

2) Feeding the new feature matrix  $Y$  to a fully connected layer. Clearly the graph convolution is the key to the huge performance gain.

Let us examine the graph convolution carefully. Suppose that we add a self-loop to each vertex in the graph, then the adjacency matrix of the new graph is  $\tilde{A} = A + I$ . The Laplacian smoothing (Taubin 1995) on each channel of the input features is defined as:

$$\hat{y}_i = (1 - \gamma) \mathbf{x}_i + \gamma \sum_j \frac{\tilde{a}_{ij}}{d_i} \mathbf{x}_j \quad (\text{for } 1 \leq i \leq n), \quad (9)$$

where  $0 < \gamma \leq 1$  is a parameter which controls the weighting between the features of the current vertex and the features of its neighbors. We can write the Laplacian smoothing in matrix form:

$$\hat{Y} = X - \gamma \tilde{D}^{-1} \tilde{L} X = (I - \gamma \tilde{D}^{-1} \tilde{L}) X, \quad (10)$$

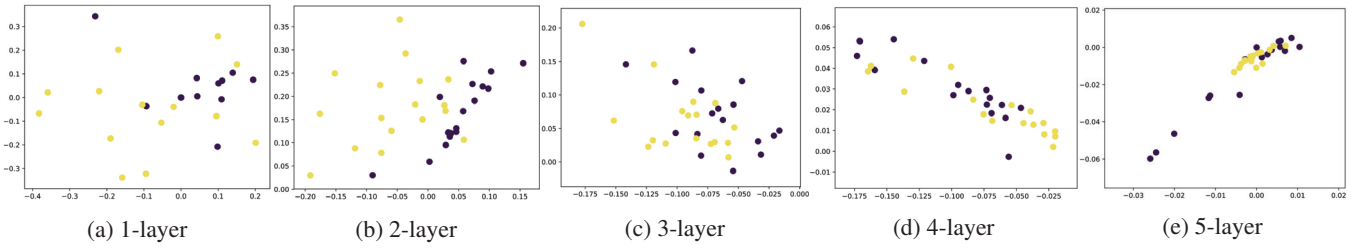


Figure 2: Vertex embeddings of Zachary’s karate club network with GCNs with 1,2,3,4,5 layers.

where  $\tilde{L} = \tilde{D} - \tilde{A}$ . By letting  $\gamma = 1$ , i.e., only using the neighbors’ features, we have  $\hat{Y} = \tilde{D}^{-1}\tilde{A}X$ , which is the standard form of Laplacian smoothing.

Now if we replace the normalized Laplacian  $\tilde{D}^{-1}\tilde{L}$  with the symmetrically normalized Laplacian  $\tilde{D}^{-\frac{1}{2}}\tilde{L}\tilde{D}^{-\frac{1}{2}}$  and let  $\gamma = 1$ , we have  $\hat{Y} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X$ , which is exactly the graph convolution in Eq. (8). We thus call the graph convolution a special form of Laplacian smoothing – symmetric Laplacian smoothing. Note that here the smoothing still includes the current vertex’s features, as each vertex has a self-loop and is its own neighbor.

The Laplacian smoothing computes the new features of a vertex as the weighted average of itself and its neighbors’. Since vertices in the same cluster tend to be densely connected, the smoothing makes their features similar, which makes the subsequent classification task much easier. As we can see from Table 1, applying the smoothing only once has already led to a huge performance gain.

**Multi-layer Structure.** We can also see from Table 1 that while the 2-layer FCN only slightly improves over the 1-layer FCN, the 2-layer GCN significantly improves over the 1-layer GCN by a large margin. This is because applying smoothing again on the activations of the first layer makes the output features of vertices in the same cluster more similar and further eases the classification task.

### When GCNs Fail

We have shown that the graph convolution is essentially a type of Laplacian smoothing. A natural question is how many convolutional layers should be included in a GCN? Certainly not the more the better. On the one hand, a GCN with many layers is difficult to train. On the other hand, repeatedly applying Laplacian smoothing may mix the features of vertices from different clusters and make them indistinguishable. In the following, we illustrate this point with a popular dataset.

We apply GCNs with different number of layers on the Zachary’s karate club dataset (Zachary 1977), which has 34 vertices of two classes and 78 edges. The GCNs are untrained with the weight parameters initialized randomly as in (Glorot and Bengio 2010). The dimension of the hidden layers is 16, and the dimension of the output layer is 2. The feature vector of each vertex is a one-hot vector. The outputs of each GCN are plotted as two-dimensional points in Fig. 2. We can observe the impact of the graph convolution (Laplacian smoothing) on this small dataset. Applying the smooth-

ing once, the points are not well-separated (Fig. 2a). Applying the smoothing twice, the points from the two classes are separated relatively well. Applying the smoothing again and again, the points are mixed (Fig. 2c, 2d, 2e). As this is a small dataset and vertices between two classes have quite a number of connections, the mixing happens quickly.

In the following, we will prove that by repeatedly applying Laplacian smoothing many times, the features of vertices within each connected component of the graph will converge to the same values. For the case of symmetric Laplacian smoothing, they will converge to be proportional to the square root of the vertex degree.

Suppose that a graph  $\mathcal{G}$  has  $k$  connected components  $\{C_i\}_{i=1}^k$ , and the indication vector for the  $i$ -th component is denoted by  $\mathbf{1}^{(i)} \in \mathbb{R}^n$ . This vector indicates whether a vertex is in the component  $C_i$ , i.e.,

$$\mathbf{1}_j^{(i)} = \begin{cases} 1, & v_j \in C_i \\ 0, & v_j \notin C_i \end{cases} \quad (11)$$

**Theorem 1.** *If a graph has no bipartite components, then for any  $\mathbf{w} \in \mathbb{R}^n$ , and  $\alpha \in (0, 1]$ ,*

$$\lim_{m \rightarrow +\infty} (I - \alpha L_{rw})^m \mathbf{w} = [\mathbf{1}^{(1)}, \mathbf{1}^{(2)}, \dots, \mathbf{1}^{(k)}] \theta_1,$$

$$\lim_{m \rightarrow +\infty} (I - \alpha L_{sym})^m \mathbf{w} = D^{-\frac{1}{2}} [\mathbf{1}^{(1)}, \mathbf{1}^{(2)}, \dots, \mathbf{1}^{(k)}] \theta_2,$$

where  $\theta_1 \in \mathbb{R}^k, \theta_2 \in \mathbb{R}^k$ , i.e., they converge to a linear combination of  $\{\mathbf{1}^{(i)}\}_{i=1}^k$  and  $\{D^{-\frac{1}{2}}\mathbf{1}^{(i)}\}_{i=1}^k$  respectively.

*Proof.*  $L_{rw}$  and  $L_{sym}$  have the same  $n$  eigenvalues (by multiplicity) with different eigenvectors (Von Luxburg 2007). If a graph has no bipartite components, the eigenvalues all fall in  $[0, 2)$  (Chung 1997). The eigenspaces of  $L_{rw}$  and  $L_{sym}$  corresponding to eigenvalue 0 are spanned by  $\{\mathbf{1}^{(i)}\}_{i=1}^k$  and  $\{D^{-\frac{1}{2}}\mathbf{1}^{(i)}\}_{i=1}^k$  respectively (Von Luxburg 2007). For  $\alpha \in (0, 1]$ , the eigenvalues of  $(I - \alpha L_{rw})$  and  $(I - \alpha L_{sym})$  all fall into  $(-1, 1]$ , and the eigenspaces of eigenvalue 1 are spanned by  $\{\mathbf{1}^{(i)}\}_{i=1}^k$  and  $\{D^{-\frac{1}{2}}\mathbf{1}^{(i)}\}_{i=1}^k$  respectively. Since the absolute value of all eigenvalues of  $(I - \alpha L_{rw})$  and  $(I - \alpha L_{sym})$  are less than or equal to 1, after repeatedly multiplying them from the left, the result will converge to the linear combination of eigenvectors of eigenvalue 1, i.e. the linear combination of  $\{\mathbf{1}^{(i)}\}_{i=1}^k$  and  $\{D^{-\frac{1}{2}}\mathbf{1}^{(i)}\}_{i=1}^k$  respectively.  $\square$

Note that since an extra self-loop is added to each vertex, there is no bipartite component in the graph. Based on

the above theorem, over-smoothing will make the features indistinguishable and hurt the classification accuracy.

The above analysis raises potential concerns about stacking many convolutional layers in a GCN. Besides, a deep GCN is much more difficult to train. In fact, the GCN used in (Kipf and Welling 2017) is a 2-layer GCN. However, since the graph convolution is a localized filter – a linear combination of the feature vectors of adjacent neighbors, a shallow GCN cannot sufficiently propagate the label information to the entire graph with only a few labels. As shown in Fig. 1, the performance of GCNs (with or without validation) drops quickly as the training size shrinks. In fact, the accuracy of GCNs decreases much faster than the accuracy of label propagation. Since label propagation only uses the graph information while GCNs utilize both structural and vertex features, it reflects the inability of the GCN model in exploring the global graph structure.

Another problem with the GCN model in (Kipf and Welling 2017) is that it requires an additional validation set for early stopping in training, which is essentially using the prediction accuracy on the validation set for model selection. If we optimize a GCN on the training data without using the validation set, it will have a significant drop in performance. As shown in Fig. 1, the performance of the GCN without validation drops much sharper than the GCN with validation. In (Kipf and Welling 2017), the authors used an additional set of 500 labeled data for validation, which is much more than the total number of training data. This is certainly undesirable as it defeats the purpose of semi-supervised learning. Furthermore, it makes the comparison of GCNs with other methods unfair as other methods such as label propagation may not need the validation data at all.

## 4 Solutions

We summarize the advantages and disadvantages of the GCN model as follows. The advantages are: 1) the graph convolution – Laplacian smoothing helps making the classification problem much easier; 2) the multi-layer neural network is a powerful feature extractor. The disadvantages are: 1) the graph convolution is a localized filter, which performs unsatisfactorily with few labeled data; 2) the neural network needs considerable amount of labeled data for validation and model selection.

We want to make best use of the advantages of the GCN model while overcoming its limits. This naturally leads to a co-training (Blum and Mitchell 1998) idea.

### Co-Train a GCN with a Random Walk Model

We propose to co-train a GCN with a random walk model as the latter can explore the global graph structure, which complements the GCN model. In particular, we first use a random walk model to find the most confident vertices – the nearest neighbors to the labeled vertices of each class, and then add them to the label set to train a GCN. Unlike in (Kipf and Welling 2017), we directly optimize the parameters of a GCN on the training set, without requiring additional labeled data for validation.

We choose to use the partially absorbing random walks (ParWalks) (Wu et al. 2012) as our random walk model. A

---

### Algorithm 1 Expand the Label Set via ParWalks

---

```

1:  $P := (L + \alpha\Lambda)^{-1}$ 
2: for each class  $k$  do
3:    $\mathbf{p} := \sum_{j \in \mathcal{S}_k} P_{:,j}$ 
4:   Find the top  $t$  vertices in  $\mathbf{p}$ 
5:   Add them to the training set with label  $k$ 
6: end for

```

---

partially absorbing random walk is a second-order Markov chain with partial absorption at each state. It was shown in (Wu, Li, and Chang 2013) that with proper absorption settings, the absorption probabilities can well capture the global graph structure. Importantly, the absorption probabilities can be computed in a closed-form by solving a simple linear system, and can be fast approximated by random walk sampling or scaled up on top of vertex-centric graph engines (Guo et al. 2017).

The algorithm to expand the training set is described in Algorithm 1. First, we calculate the normalized absorption probability matrix  $P = (L + \alpha\Lambda)^{-1}$  (the choice of  $\Lambda$  may depend on data).  $P_{i,j}$  is the probability of a random walk from vertex  $i$  being absorbed by vertex  $j$ , which represents how likely  $i$  and  $j$  belong to the same class. Second, we need to measure the confidence of a vertex belonging to class  $k$ . We partition the labeled vertices into  $\mathcal{S}_1, \mathcal{S}_2, \dots$ , where  $\mathcal{S}_k$  denotes the set of labeled data of class  $k$ . For each class  $k$ , we calculate a confidence vector  $\mathbf{p} = \sum_{j \in \mathcal{S}_k} P_{:,j}$ , where  $\mathbf{p} \in \mathbb{R}^n$  and  $p_i$  is the confidence of vertex  $i$  belonging to class  $k$ . Finally, we find the  $t$  most confident vertices and add them to the training set with label  $k$  to train a GCN.

### GCN Self-Training

Another way to make a GCN “see” more training examples is to self-train a GCN. Specifically, we first train a GCN with given labels, then select the most confident predictions for each class by comparing the softmax scores, and add them to the label set. We then continue to train the GCN with the expanded label set, using the pre-trained GCN as initialization. This is described in Algorithm 2.

The most confident instances found by the GCN are supposed to share similar (but not the same) features with the labeled data. Adding them to the labeled set will help training a more robust and accurate classifier. Furthermore, it complements the co-training method in the situation that a graph has many isolated small components and it is not possible to propagate labels with random walks.

---

### Algorithm 2 Expand the Label Set via Self-Training

---

```

1:  $\mathbf{Z} := GCN(X) \in \mathbb{R}^{n \times F}$ , the output of GCN
2: for each class  $k$  do
3:   Find the top  $t$  vertices in  $Z_{i,k}$ 
4:   Add them to the training set with label  $k$ 
5: end for

```

---

**Combine Co-Training and Self-Training.** To improve

the diversity of labels and train a more robust classifier, we propose to combine co-training and self-learning. Specifically, we expand the label set with the most confident predictions found by the random walk and those found by the GCN itself, and then use the expanded label set to continue to train the GCN. We call this method “Union”. To find more accurate labels to add to the labeled set, we also propose to add the most confident predictions found by both the random walk and the GCN. We call this method “Intersection”.

Note that we optimize all our methods on the expanded label set, without requiring any additional validation data. As long as the expanded label set contains enough correct labels, our methods are expected to train a good GCN classifier. But how much labeled data does it require to train a GCN? Suppose that the number of layers of the GCN is  $\tau$ , and the average degree of the underlying graph is  $\hat{d}$ . We propose to estimate the lower bound of the number of labels  $\eta = |\mathcal{V}_l|$  by solving  $(\hat{d})^\tau * \eta \approx n$ . The rationale behind this is to estimate how many labels are needed to for a GCN with  $\tau$  layers to propagate them to cover the entire graph.

## 5 Experiments

In this section, we conduct extensive experiments on real benchmarks to verify our theory and the proposed methods, including Co-Training, Self-Training, Union, and Intersection (see Section 4).

We compare our methods with several state-of-the-art methods, including GCN with validation (GCN+V); GCN without validation (GCN-V); GCN with Chebyshev filter (Cheby) (Kipf and Welling 2017); label propagation using ParWalks (LP) (Wu et al. 2012); Planetoid (Yang, Cohen, and Salakhutdinov 2016); DeepWalk (Perozzi, Al-Rfou, and Skiena 2014); manifold regularization (ManiReg) (Belkin, Niyogi, and Sindhvani 2006); semi-supervised embedding (SemiEmb) (Weston et al. 2008); iterative classification algorithm (ICA) (Sen et al. 2008).

### Experimental Setup

We conduct experiments on three commonly used citation networks: CiteSeer, Cora, and PubMed (Sen et al. 2008). The statistics of the datasets are summarized in Table 2. On each dataset, a document is described by a bag-of-words feature vector, i.e., a 0/1-valued vector indicating the absence/presence of a certain word. The citation links between documents are described by a 0/1-valued adjacency matrix. The datasets we use for testing are provided by the authors of (Yang, Cohen, and Salakhutdinov 2016) and (Kipf and Welling 2017).

For ParWalks, we set  $\Lambda = I$ , and  $\alpha = 10^{-6}$ , following Wu et al.. For GCNs, we use the same hyper-parameters as in (Kipf and Welling 2017): a learning rate of 0.01, 200 maximum epochs, 0.5 dropout rate,  $5 \times 10^{-4}$   $L2$  regularization weight, 2 convolutional layers, and 16 hidden units, which are validated on Cora by Kipf and Welling. For each run, we randomly split labels into a small set for training, and a set with 1000 samples for testing. For GCN+V, we follow (Kipf and Welling 2017) to sample additional 500 labels for validation. For GCN-V, we simply optimize the GCN using

Table 2: Dataset statistics

Dataset	Nodes	Edges	Classes	Features
CiteSeer	3327	4732	6	3703
Cora	2708	5429	7	1433
PubMed	19717	44338	3	500

training accuracy. For Cheby, we set the polynomial degree  $K = 2$  (see Eq. (1)). We test these methods with 0.5%, 1%, 2%, 3%, 4%, 5% training size on Cora and CiteSeer, and with 0.03%, 0.05%, 0.1%, 0.3% training size on PubMed. We choose these labeling rates for easy comparison with (Kipf and Welling 2017), (Yang, Cohen, and Salakhutdinov 2016), and other methods. We report the mean accuracy of 50 runs except for the results on PubMed (Yang, Cohen, and Salakhutdinov 2016), which are averaged over 10 runs.

### Results Analysis

The classification results are summarized in Table 3, 4 and 5, where the highest accuracy in each column is highlighted in bold and the top 3 are underlined. Our methods are displayed at the bottom half of each table.

We can see that the performance of Co-Training is closely related to the performance of LP. If the data has strong manifold structure, such as PubMed, Co-Training performs the best. In contrast, Self-Training is the worst on PubMed, as it does not utilize the graph structure. But Self-Training does well on CiteSeer where Co-Training is overall the worst. Intersection performs better when the training size is relatively large, because it filters out many labels. Union performs best in many cases since it adds more diverse labels to the training set.

**Comparison with GCNs.** At a glance, we can see that on each dataset, our methods outperform others by a large margin in most cases. When the training size is small, all our methods are far better than GCN-V, and much better than GCN+V in most cases. For example, with labeling rate 1% on Cora and CiteSeer, our methods improve over GCN-V by 23% and 28%, and improve over GCN+V by 12% and 7%. With labeling rate 0.05% on PubMed, our methods improve over GCN-V and GCN+V by 37% and 18% respectively. This verifies our analysis that the GCN model cannot effectively propagate labels to the entire graph with small training size. When the training size grows, our methods are still better than GCN+V in most cases, demonstrating the effectiveness of our approaches. When the training size is large enough, our methods and GCNs perform similarly, indicating that the given labels are sufficient for training a good GCN classifier. Cheby does not perform well in most cases, which is probably due to overfitting.

**Comparison with other methods.** We compare our methods with other state-of-the-art methods in Table 6. The experimental setup is the same except that for every dataset, we sample 20 labels for each class, which corresponds to the total labeling rate of 3.6% on CiteSeer, 5.1% on Cora, and 0.3% on PubMed. The results of other baselines are copied from (Kipf and Welling 2017). Our methods perform similarly as GCNs and outperform other baselines significantly.

Table 3: Classification Accuracy On Cora

Label Rate	Cora					
	0.5%	1%	2%	3%	4%	5%
LP	56.4	62.3	65.4	67.5	69.0	70.2
Cheby	38.0	52.0	62.4	70.8	74.1	77.6
GCN-V	42.6	56.9	67.8	74.9	77.6	79.3
GCN+V	50.9	62.3	72.2	76.5	78.4	79.7
Co-training	56.6	66.4	73.5	75.9	78.9	80.8
Self-training	53.7	66.1	73.8	77.2	79.4	80.0
Union	<b>58.5</b>	<b>69.9</b>	<b>75.9</b>	<b>78.5</b>	<b>80.4</b>	<b>81.7</b>
Intersection	49.7	65.0	72.9	77.1	79.4	80.2

Table 4: Classification Accuracy on CiteSeer

Label Rate	CiteSeer					
	0.5%	1%	2%	3%	4%	5%
LP	34.8	40.2	43.6	45.3	46.4	47.3
Cheby	31.7	42.8	59.9	66.2	68.3	69.3
GCN-V	33.4	46.5	62.6	66.9	68.4	69.5
GCN+V	43.6	55.3	64.9	67.5	68.7	69.6
Co-training	<b>47.3</b>	55.7	62.1	62.5	64.5	65.5
Self-training	43.3	58.1	68.2	69.8	70.4	71.0
Union	46.3	<b>59.1</b>	66.7	66.7	67.6	68.2
Intersection	42.9	<b>59.1</b>	<b>68.6</b>	<b>70.1</b>	<b>70.8</b>	<b>71.2</b>

Although we did not directly compare with other baselines, we can see from Table 3, 4 and 5 that our methods with much fewer labels already outperform many baselines. For example, our method Union on Cora (Table 3) with 2% labeling rate (54 labels) beats all other baselines with 140 labels (Table 6).

**Influence of the Parameters.** A common parameter of our methods is the number of newly added labels. Adding too many labels will introduce noise, but with too few labels we cannot train a good GCN classifier. As described in the end of Section 4, we can estimate the lower bound of the total number of labels  $\eta$  needed to train a GCN by solving  $(\hat{d})^\tau * \eta \approx n$ . We use  $3\eta$  in our experiments. Actually, we found that  $2\eta$ ,  $3\eta$  and  $4\eta$  perform similarly in the experiments. We follow Kipf and Welling to set the number of convolutional layers as 2. We also observed in the experiments that 2-layer GCNs performed the best. When the number of convolutional layers grows, the classification accuracy decreases drastically, which is probably due to overfitting.

**Computational Cost.** For Co-Training, the overhead is the computational cost of the random walk model, which requires solving a sparse linear system. In our experiments, the time is negligible on Cora and CiteSeer as there are only a few thousand vertices. On PubMed, it takes less than 0.38 seconds in MatLab R2015b. As mentioned in Section 4, the computation can be further speeded up using vertex-centric graph engines (Guo et al. 2017), so the scalability of our method is not an issue. For Self-Training, we only need to run a few epochs in addition to training a GCN. It converges fast as it builds on a pre-trained GCN. Hence, the running time of Self-Training is comparable to a GCN.

Table 5: Classification Accuracy On PubMed

Label Rate	PubMed			
	0.03%	0.05%	0.1%	0.3%
LP	61.4	66.4	65.4	66.8
Cheby	40.4	47.3	51.2	72.8
GCN-V	46.4	49.7	56.3	76.6
GCN+V	60.5	57.5	65.9	77.8
Co-training	<b>62.2</b>	<b>68.3</b>	<b>72.7</b>	<b>78.2</b>
Self-training	51.9	58.7	66.8	77.0
Union	58.4	64.0	70.7	<b>79.2</b>
Intersection	52.0	59.3	69.4	77.6

Table 6: Accuracy under 20 Labels per Class

Method	CiteSeer	Cora	Pubmed
ManiReg	60.1	59.5	70.7
SemiEmb	59.6	59.0	71.7
LP	45.3	68.0	63.0
DeepWalk	43.2	67.2	65.3
ICA	69.1	75.1	73.9
Planetoid	64.7	75.7	77.2
GCN-V	68.1	80.0	78.2
GCN+V	68.9	80.3	<b>79.1</b>
Co-training	64.0	79.6	77.1
Self-training	67.8	80.2	76.9
Union	65.7	<b>80.5</b>	78.3
Intersection	<b>69.9</b>	79.8	77.0

## 6 Conclusions

Understanding deep neural networks is crucial for realizing their full potentials in real applications. This paper contributes to the understanding of the GCN model and its application in semi-supervised classification. Our analysis not only reveals the mechanisms and limitations of the GCN model, but also leads to new solutions overcoming its limits. In future work, we plan to develop new convolutional filters which are compatible with deep architectures, and exploit advanced deep learning techniques to improve the performance of GCNs for more graph-based applications.

## Acknowledgments

This research received support from the grant 1-ZVJJ funded by the Hong Kong Polytechnic University. The authors would like to thank the reviewers for their insightful comments and useful discussions.

## References

- Belkin, M.; Niyogi, P.; and Sindhwani, V. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research* 7:2434.
- Bengio, Y.; Delalleau, O.; and Le Roux, N. 2006. Label propagation and quadratic criterion. *Semi-supervised learning* 193–216.

- Blum, A., and Chawla, S. 2001. Learning from labeled and unlabeled data using graph mincuts. In *ICML*, 19–26. ACM.
- Blum, A., and Mitchell, T. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th annual conference on Computational learning theory*, 92–100. ACM.
- Blum, A.; Lafferty, J.; Rwebangira, M.; and Reddy, R. 2004. Semi-supervised learning using randomized mincuts. In *ICML*, 13. ACM.
- Bronstein, M. M.; Bruna, J.; LeCun, Y.; Szlam, A.; and Vandergheynst, P. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34(4):18–42.
- Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2014. Spectral networks and locally connected networks on graphs. *ICLR*.
- Chapelle, O., and Zien, A. 2005. Semi-supervised classification by low density separation. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, 57–64. Max-Planck-Gesellschaft.
- Chung, F. R. 1997. *Spectral graph theory*. American Mathematical Soc.
- Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 3844–3852. Curran Associates, Inc.
- Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256. PMLR.
- Guo, H.; Tang, R.; Ye, Y.; Li, Z.; and He, X. 2017. A graph-based push service platform. In *International Conference on Database Systems for Advanced Applications*, 636–648. Springer.
- Hammond, D. K.; Vandergheynst, P.; and Gribonval, R. 2011. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis* 30(2):129–150.
- Joachims, T. 2003. Transductive learning via spectral graph partitioning. In *ICML*, 290–297. ACM.
- Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
- Lake, B. M.; Salakhutdinov, R.; and Tenenbaum, J. B. 2015. Human-level concept learning through probabilistic program induction. *Science* 350(6266):1332–1338.
- Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710. ACM.
- Rasmus, A.; Berglund, M.; Honkala, M.; Valpola, H.; and Raiko, T. 2015. Semi-supervised learning with ladder networks. In *NIPS*, 3546–3554. Curran Associates, Inc.
- Rezende, D.; Danihelka, I.; Gregor, K.; Wierstra, D.; et al. 2016. One-shot generalization in deep generative models. In *ICML 2016*, 1521–1529. ACM.
- Sandryhaila, A., and Moura, J. M. 2013. Discrete signal processing on graphs. *IEEE transactions on signal processing* 61(7):1644–1656.
- Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine* 29(3):93.
- Shuman, D. I.; Narang, S. K.; Frossard, P.; Ortega, A.; and Vandergheynst, P. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* 30(3):83–98.
- Talukdar, P. P., and Crammer, K. 2009. New regularized algorithms for transductive learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 442–457. Springer.
- Taubin, G. 1995. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 351–358. ACM.
- Von Luxburg, U. 2007. A tutorial on spectral clustering. *Statistics and computing* 17(4):395–416.
- Weston, J.; Ratle, F.; Mobahi, H.; and Collobert, R. 2008. Deep learning via semi-supervised embedding. In *Proceedings of the 25th international conference on Machine learning*, 1168–1175. ACM.
- Wu, X.; Li, Z.; So, A. M.; Wright, J.; and Chang, S.-f. 2012. Learning with Partially Absorbing Random Walks. In *NIPS* 25, 3077–3085. Curran Associates, Inc.
- Wu, X.-M.; Li, Z.; and Chang, S. 2013. Analyzing the harmonic structure in graph-based learning. In *NIPS*.
- Yang, Z.; Cohen, W. W.; and Salakhutdinov, R. 2016. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International conference on Machine learning*, 40–48. ACM.
- Zachary, W. W. 1977. An information flow model for conflict and fission in small groups. *Journal of anthropological research* 33(4):452–473.
- Zhou, D.; Bousquet, O.; Lal, T. N.; Weston, J.; and Schölkopf, B. 2004. Learning with local and global consistency. In *NIPS* 16, 321–328. MIT Press.
- Zhu, X., and Goldberg, A. 2009. Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 3(1):1–130.
- Zhu, X.; Ghahramani, Z.; and Lafferty, J. D. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML 2003*, 912–919. ACM.