Directed Graph Convolutional Network

Zekun Tong
National University of Singapore
zekuntong@u.nus.edu

Yuxuan Liang
National University of Singapore
yuxliang@outlook.com

Changsheng Sun
National University of Singapore
changsheng_sun@outlook.com

David S. Rosenblum
National University of Singapore
david@comp.nus.edu.sg

Andrew Lim*
National University of Singapore
isealim@nus.edu.sg

arXiv:2004.13970v1 [cs.LG] 29 Apr 2020

ABSTRACT

Graph Convolutional Networks (GCNs) have been widely used due to their outstanding performance in processing graph-structured data. However, the undirected graphs limit their application scope. In this paper, we extend spectral-based graph convolution to directed graphs by using first- and second-order proximity, which can not only retain the connection properties of the directed graph, but also expand the receptive field of the convolution operation. A new GCN model, called DGCN, is then designed to learn representations on the directed graph, leveraging both the first- and second-order proximity information. We empirically show the fact that GCNs working only with DGCNs can encode more useful information from graph and help achieve better performance when generalized to other models. Moreover, extensive experiments on citation networks and co-purchase datasets demonstrate the superiority of our model against the state-of-the-art methods.

KEYWORDS

Graph Neural Networks; Semi-Supervised Learning; Proximity

1 INTRODUCTION

Graph structure is a common form of data. Graphs have a very strong ability to represent complex structures and can easily express entities and their relationships. Graph Convolutional Networks (GCNs) [4, 7, 8, 11, 27, 33] are a CNNs variant on graphs and effectively learns underlying pairwise relations among data vertices. We can divide GCNs into two categories: spectral-based [4, 11, 15] and spatial-based [7, 27]. A representative structure of spectral-based GCNs [11] has multiple layers that stack first-order spectral filters and learn graph representations using a nonlinear activation function, while spatial-based approaches design neighborhood features aggregating schemes to achieve graph convolutions [32, 35]. Various GCNs have significant improvements in benchmark datasets [7, 11, 27]. Such breakthroughs have promoted the exploration of variant networks: the Attention-based Graph Neural Network(AGNN) [26] uses the attention mechanism to replace the propagation layers in order to learn dynamic neighborhood information; the FastGCN [3] interprets graph convolutions as integral transforms of embedding functions while using batch training to speed up. The positive effects of different GCN variants greatly promote their use in various task fields, including but not limited to social networks [3], quantum chemistry [16], text classification [34] and image recognition [30].

One of the main reasons that a GCN can achieve such good results on a graph is that makes full use of the structure of the graph. It captures rich information from the neighborhood of object nodes through the links, instead of being limited to a specific distance range. General GCN models provide a neighborhood aggregation scheme for each node to gain a representation vector, and then learn a mapping function to predict the node attributes [9]. Since spatial-based methods need to traverse surrounding nodes when aggregating features, they usually add significant overhead to computation and memory usage [31]. On the contrary, spectral-based methods use matrix multiplication instead of traversal search, which greatly improves the training speed. Thus, in this paper, we focus mainly on the spectral-based method.

Although the above methods have achieved improvement in many aspects, there are two major shortcomings with existing spectral-based GCN methods.

First, spectral-based GCNs are limited to apply to undirected graphs [32]. For directed graphs, the only way is to relax the graph structure from a directed to an undirected graph by symmetrizing the adjacency matrix. In this way we can get the semi-definite Laplacian matrix, but at the same time we also lose the unique structure of the directed graph. For example, in a citation graph, later published articles can cite former published articles, but the reverse is not true. This is a unique time series relationship. If we transform it into an undirected graph, we lose this part of the information. Although we can represent the original directed graph in another form, such a temporal graph learned by combination of Recurrent Neural Networks (RNNs) and GCNs [20], we still want to dig more structural features from the original without adding extra components.

Second, in most existing spectral-based GCN models, during each convolution operation, only 1-hop node features are taken into account (using the same adjacency matrix), which means they only capture the *first-order* information between vertices. It is natural to consider direct links when extracting local features, but this is not always the case. In some real-world graph data, many legitimate relationships may not be encoded via first-order links [25]. For instance, people in social networking communities share common interests, but they don not necessarily contact each other. In other words, the features we get by *first-order* proximity are likely to be insufficient. Although we can obtain more information by stacking multiple layers of GCNs, multi-layer networks will introduce more trainable parameters, making them prone to overfitting when the label rate is small or needing extra labeled information, which is not label efficient [14]. Therefore, we need a more efficient feature extraction method.

*Corresponding author

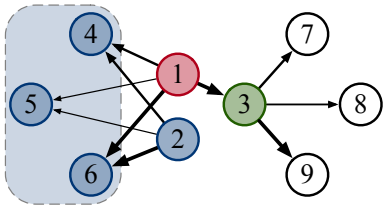


Figure 1: A simple weighted directed graph example. Line width indicates the weight of the edges. The node v_1 has *first-order* proximity with v_3. It also has *second-order* proximity with v_2, because they have shared neighbors $\{v_4, v_5, v_6\}$. Both v_2 and v_3's features should be considered when aggregating v_1's feature.

To address these issues, we leverage *second-order* proximity between vertices as a complement to existing methods, which inspire from the *hub* and *authority* web model [12, 37]. By using *second-order* proximity, the directional features of the directed graph can be retained. Additionally, the receptive field of the graph convolution can be expanded to extract more features. Different from *first-order* proximity, judging whether two nodes have *second-order* proximity does not require nodes to have paths between them, as long as they share common neighbors, which can be considered as *second-order* proximity. In other words, nodes with more shared neighbors have stronger *second-order* proximity in the graph. This general notion also appears in sociology [6], psychology [22] and daily life: people who have a lot of common friends are more likely to be friends. A simple example is shown in Figure 1. When considering the neighborhood feature aggregation of v_1 in the $layer_1$, we need to consider v_3, because it has a *first-order* connection with v_1, but we also need to aggregate v_2's features, due to the high *second-order* similarity with v_1 in sharing three common neighbors.

In this paper, we present a new spectral-based model on directed graphs, **D**irected **G**raph **C**onvolutional **N**etworks (**DGCN**), which utilizes *first & second-order* proximity to extract graph information. We not only consider basic *first-order* proximity to obtain neighborhood information, but also take *second-order* proximity into account, which is a complement for sparsity of *first-order* information in real-world data. What's more, we verify this efficiency by extending *Feature and Label Smoothness* measurements [9] to our application scope. Through experiments, we empirically show that our model exhibits superior performance against baselines while the number of parameters and computational consumption are significantly reduced.

In summary, our study has the following contributions:

(1) We present a novel graph convolutional networks called the **"DGCN"**, which can be applied to the directed graphs by utilizing *first- and second-order* proximity. To our knowledge, this is the first-ever attempt that enables the spectral-based GCNs to generalize to directed graphs.
(2) We define *first- and second-order* proximity on the directed graph, which are designed for expanding the convolution operation receptive field, extracting and leveraging graph information. Meanwhile, we empirically show that this method has both feature and label efficiency, while also demonstrating powerful generalization capability.
(3) We experiment with semi-supervised classification tasks on various real-world datasets, which validates the effectiveness of *first- and second-order* proximity and the improvements obtained by DGCNs over other models. We will release our code for public use soon.

2 PRELIMINARIES

In this section, we first clarify the terminology and preliminary knowledge of our study, and then present our task in detail. Particularly, we use bold capital letters (*e.g.,* $\mathbf{X}$) and bold lowercase letters (*e.g.,* $\mathbf{x}$) to denote matrices and vectors, respectively. We use non-bold letters (*e.g.,* x) to represent scalars and Greek letters (*e.g.,* λ) as parameters.

2.1 Formulation

Definition 1. ***Directed Graph*** *[21]. A general graph has n vertices or nodes is define as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = (v_1, ..., v_n)$ is vertex set and $\mathcal{E} \subset \{1, .., n\} \times \{1, ..., n\}$ is edge set. Each edge $e \in \mathcal{E}$ is an ordered pair $e = (i, j)$ between vertex v_i and v_j. If any of its edges e is associated with a weight $W_{i,j} > 0$, the graph is weighted. When pair $(i, j) \notin \mathcal{E}$, we set $W_{i,j} = 0$. A graph is directed when has $(u, v) \not\equiv (v, u)$ and $w_{i,j} \not\equiv w_{j,i}$.*

Here, we consider the undirected graph as a directed graph has two directed edges with opposite directions and equal weights, and binary graph as a weighted graph which edge weight values only take from 0 or 1. Besides, we only consider non-negative weights.

Furthermore, to measure our model's ability to extract surrounding information, we define two indicators[9, 29] on the directed graph: **Feature Smoothness**, which is used to evaluate how much surrounding information we can obtain and **Label Smoothness** for evaluating how useful the obtained information is.

Definition 2. ***First & second-order edges in directed graph.*** *Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For an order pair (i, j), where i and $j \in \mathcal{V}$. If $(i, j) \in \mathcal{E}$, we say that order pair (i, j) is first-order edge. If it exists any vertex $k \in \mathcal{V}$ that satisfies order pairs (k, i) and $(k, j) \in \mathcal{E}$ or (i, k) and $(j, k) \in \mathcal{E}$, we define order pair (i, j) as second-order edge. The edge set has both first & second-order edge of $\mathcal{G}$ denoted by $\mathcal{E}'$.*

Definition 3. ***Feature Smoothness.*** *We define the Feature Smoothness λ_f over normalized node feature space $\mathcal{X} = [0, 1]^d$ as follows,*

$$\lambda_f = \frac{\left\| \sum_{i \in \mathcal{V}'} \left(\sum_{e_{i,j} \in \mathcal{E}'} (x_i - x_j)^2 \right) \right\|_1}{|\mathcal{E}'| \cdot d}, \tag{1}$$

where $||\cdot||_1$ is the Manhattan norm, d is the node feature dimension and x_i is feature of vertex i.

Definition 4. ***Label Smoothness.*** *For the node classification task, we want to determine how useful the information obtained from the surrounding nodes is. We believe that the information obtained is valid when the current node label is consistent with the surrounding node labels, otherwise it is invalid. Based on this, we define Label Smoothness λ_l as:*

$$\lambda_l = \frac{\sum_{e_{i,j} \in \mathcal{E}'} \left(\mathbb{I}\left(v_i \simeq v_j\right) \right)}{|\mathcal{E}'|}, \tag{2}$$

where $\mathbb{I}(\cdot)$ is an indictor function and we define $v_i \simeq v_j$ if the label of vertex i and j are the same.

2.2 Problem Statement

After giving the above formulation, we are ready to define our task.

Definition 5. ***Semi-Supervised Node Classification**[1]. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix A, and node feature matrix $\mathbf{X} \in \mathbb{R}^{N \times C}$, where $N = |\mathcal{V}|$ is the number of nodes and C is the feature dimension. Given a subset of nodes $\mathcal{V}_L \subset \mathcal{V}$, where nodes in $\mathcal{V}_L$ have observed labels and generally $|\mathcal{V}_L| << |\mathcal{V}|$. The task is using the labeled subset $\mathcal{V}_L$, node feature matrix X and adjacency matrix A predict the unknown label in $\mathcal{V}_{UL} = \mathcal{V} \setminus \mathcal{V}_L$.*

3 UNDIRECTED GRAPH CONVOLUTION

In this section, we will go through the spectral graph convolutions defined on the undirected graph.

We have the undirected graph Laplacian $\mathbf{L}' = \mathbf{D}' - \mathbf{A}'$, $\mathbf{A}'$ is the adjacency matrix of the graph, $\mathbf{D}'$ is a diagonal matrix of node degree, $D'_{ii} = \sum_j (A'_{i,j})$. $\mathbf{L}'$ can be factored as $\mathbf{L}' = \mathbf{I} - \mathbf{D}'^{-\frac{1}{2}}\mathbf{A}'\mathbf{D}'^{-\frac{1}{2}} = \mathbf{U}'\mathbf{\Lambda}\mathbf{U}'^T$, where $\mathbf{I}$ is identity matrix, $\mathbf{U}'$ is the matrix of eigenvectors and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues. The spectral convolutions on graph is defined as the multiplication of node feature vector $\mathbf{x} \in \mathbb{R}^N$ with a filter $\mathbf{g} \in \mathbb{R}^N$ in the Fourier domain:

$$x * g = \mathbf{U}' \left(\mathbf{U}'^T x \odot \mathbf{U}'^T g \right) = \mathbf{U}' g_\theta \mathbf{U}'^T x, \tag{3}$$

where $g_\theta = diag(\mathbf{U}^T \mathbf{x})$, $*$ represents convolution operation and $\odot$ is the element-wise Hadamard product.

Graph convolutions can be further approximated by k^{th} Chebyshev polynomials to reduce computation-consuming:

$$\mathbf{x} * \mathbf{g}_\theta \approx \sum_{i=1}^{K} \theta_i T_i(\tilde{L'}) x, \tag{4}$$

where $\tilde{\mathbf{L}'} = 2\mathbf{L}'/\lambda_{\max} - \mathbf{I}$, $\lambda_{\max}$ denotes the largest eigenvalue of L' and $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$. The K-polynomial filter $\mathbf{g}_\theta$ shows its good localization in the vertex domain through integrating the node features within the K-hop neighborhood[36]. Our model obtains node features in a different way, which will be explained in the Section 4.4.

Kipf et al. employ Graph Convolutional Networks (GCNs)[11], which is a first-order approximation of ChebNet. They assume $k = 1$, $\lambda_{max} = 2$, $\theta_0 = 2\theta$ and $\theta_1 = -\theta$ in Equation 4 to simplify the convolution process and get the following GCN convolution operation:

$$\mathbf{x} * \mathbf{g}_\theta \approx \theta \left(\mathbf{I} + \mathbf{D}'^{-\frac{1}{2}} \mathbf{A}' \mathbf{D}'^{-\frac{1}{2}} \right) \mathbf{x}. \tag{5}$$

They also use renormalization trick converting $\mathbf{I} + \mathbf{D}'^{-1/2}\mathbf{A}'\mathbf{D}'^{-1/2}$ to $\tilde{\mathbf{D}}'^{-1/2}\tilde{\mathbf{A}}'\tilde{\mathbf{D}}'^{-1/2}$, where $\tilde{\mathbf{A}}' = \mathbf{A}' + \mathbf{I}$ and $\tilde{D'_{ii}} = \sum_j (\tilde{A'}_{i,j})$, to alleviate numerical instabilities and exploding/vanishing gradients problems. The final graph convolution layer is defined as follows:

$$\mathbf{Z}' = \left(\tilde{\mathbf{D}}'^{-\frac{1}{2}} \tilde{\mathbf{A}}' \tilde{\mathbf{D}}'^{-\frac{1}{2}} \right) \mathbf{X}\Theta. \tag{6}$$

Here, $\mathbf{X} \in \mathbb{R}^{N \times C}$ is the C-dimensional node feature vector, $\Theta \in \mathbb{R}^{C \times F}$ is the filter parameters matrix and $\mathbf{Z}' \in \mathbb{R}^{N \times F}$ is the convolved result with F output dimension.

However, these derivations are based on the premise that Laplacian matrices are undirected graphs. The way they use to deal with directed graph is to relax it into an undirected graph, thereby constructing a symmetric Laplacian matrix. Although the relaxed matrix can be used for spectral convolution, it is not able to represent the actual structure of the directed graph. For instance, as we mention in the Section 1, there is a time limit for citations: previously published papers cannot cite later ones. If the citation network is relaxed to an undirected graph, this restriction will no longer exist.

In order to solve this problem, we propose a spectral-base GCN model for directed graph that leverages First- and Second-order Proximity in the following sections.

4 THE NEW MODEL: DGCN

In this section, we present our spectral-based GCN model $f(\mathbf{X}, \mathbf{A})$ for directed graphs that leverages the First- and Second-Order Proximity, called **DGCN**. We provide the mathematical motivation of directed graph convolution and consider a multi-layer Graph Convolutional Network which has the following layer-wise propagation rule, where

$$\begin{aligned} \hat{\mathbf{A}}_\mathbf{F} &= \tilde{\mathbf{D}}_\mathbf{F}^{-\frac{1}{2}} \tilde{\mathbf{A}}_\mathbf{F} \tilde{\mathbf{D}}_\mathbf{F}^{-\frac{1}{2}} \\ \hat{\mathbf{A}}_{\mathbf{S}_{\text{in}}} &= \tilde{\mathbf{D}}_{\mathbf{S}_{\text{in}}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{\mathbf{S}_{\text{in}}} \tilde{\mathbf{D}}_{\mathbf{S}_{\text{in}}}^{-\frac{1}{2}} \\ \hat{\mathbf{A}}_{\mathbf{S}_{\text{out}}} &= \tilde{\mathbf{D}}_{\mathbf{S}_{\text{out}}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{\mathbf{S}_{\text{out}}} \tilde{\mathbf{D}}_{\mathbf{S}_{\text{out}}}^{-\frac{1}{2}}, \end{aligned}$$

and

$$\mathbf{H}^{(l+1)} = \Gamma \left(\sigma(\hat{\mathbf{A}}_\mathbf{F} \mathbf{H}^{(l)} \Theta^{(l)}), \sigma(\hat{\mathbf{A}}_{\mathbf{S}_{\text{in}}} \mathbf{H}^{(l)} \Theta^{(l)}), \sigma(\hat{\mathbf{A}}_{\mathbf{S}_{\text{out}}} \mathbf{H}^{(l)} \Theta^{(l)}) \right). \tag{7}$$

Here, $\tilde{\mathbf{A}}_\mathbf{F}$ is the normalized First-order Proximity matrix with self-loop and $\tilde{\mathbf{A}}_{\mathbf{S}_{\text{in}}}, \tilde{\mathbf{A}}_{\mathbf{S}_{\text{out}}}$ are the normalized Second-order Proximity matrix with self-loop, which are defined in Section 4.1. $\Gamma(\cdot)$ is a fusion function combines the proximity matrices together defined in Section 4.2. $\Theta^{(l)}$ is a shared trainable weight matrix and $\sigma(\cdot)$ is an activation function. $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times C}$ is the matrix of activation in the l^{th} layer and $\mathbf{H}^{(0)} = \mathbf{X}$. Finally, we present our model using directed graph convolution for semi-supervised learning in detail.

4.1 First- and Second-order Proximity

To conduct the feature extraction, we not only obtain the node's features from its directly adjacent nodes, but also extract the hidden information from *second-order* neighbor nodes. Different from other methods considering K-hop neighborhood information[1, 4], we define new *First- and Second-order Proximity* and show schematically descriptions in Figure 2.

4.1.1 First-order Proximity. The first-order proximity refers to the local pairwise proximity between the vertices in a graph.

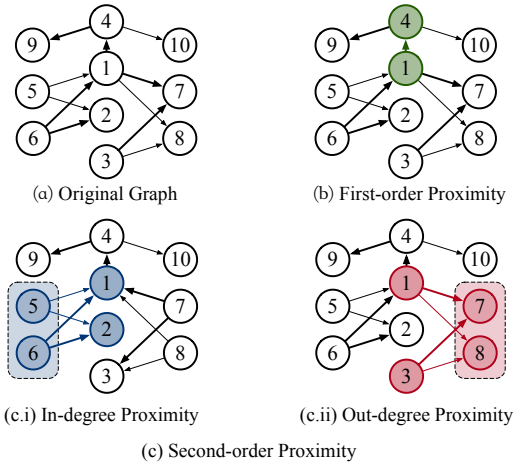


Figure 2: First- and second-order proximity examples in a weighted directed graph.

Definition 6. ***First-order Proximity Matrix.*** *In order to model the first-order proximity, we define the* first-order *proximity A_F between vertex v_i and v_j for each edge (i, j) in the graph as follows:*

$$A_F(i,j) = A^{sym}(i,j), \tag{8}$$

where $\mathbf{A}^{sym}$ is the symmetric matrix of edge weight matrix $\mathbf{A}$. If there is no edge from v_i to v_j or v_j to v_i, $A_F(i,j)$ is equal to 0.

In Figure 2(b), it is easy to find that vertex v_1 has *first-order* proximity with v_4. Note that the *first-order* proximity is relaxed for directed graphs. We use the symmetric matrix to replace to original one, which is inevitable losing some directed informations. For this part of the missing information, we will use another way to retain it, which is the *second-order* proximity. This problem does not exist for undirected graphs because its weights matrix is symmetric.

4.1.2 Second-order Proximity. The second-order proximity assumes that if two vertices share many common neighbors tend to be similar. In this case, we need to build a second-order proximity matrix, so that similar nodes can be connected with each other.

Definition 7. ***Second-order Proximity Matrix.*** *The second-order proximity between vertices is determined by the normalized weights summation of edges linked with their shared neighborhood nodes. In a directed graph $\mathcal{G}$, for vertex v_i and v_j, we define the* second-order *in-degree proximity $A_{S_{in}}(i,j)$ and out-degree proximity $A_{S_{out}}(i,j)$:*

$$A_{S_{in}}(i,j) = \sum_k \frac{A_{k,i}A_{k,j}}{\sum_v A_{k,v}} \tag{9}$$

and

$$A_{S_{out}}(i,j) = \sum_k \frac{A_{i,k}A_{j,k}}{\sum_v A_{v,k}}, \tag{10}$$

where $\mathbf{A}$ is is the weighted adjacency matrix of $\mathcal{G}$ and $k, v \in \mathcal{V}$.

Since $A_{S_{in}}(i,j)$ sums up the normalized weights of edges which array to both v_i and v_j, i.e. $\sum_k A\{i \leftarrow k \rightarrow j\}$, it can best reflect the similarity of the in-degree between vertex v_i and v_j. The larger the $A_{S_{in}}(i,j)$, the higher the similarity of the second-order in-degree.

Similarly, $A_{S_{out}}(i,j)$ measures the second-order out-degree proximity by accumulating the weights of edges from both v_i and v_j, i.e. $\sum_k A\{i \rightarrow k \leftarrow j\}$. If no shared vertices linked from/to v_i and v_j, we set their second-order proximity as 0.

A visualization of these two proximities are shown in Figure 2(c). In Figure 2.c.i, v_1 and v_2 have second-order in-degree proximity, because they share common neighbors $\{1 \leftarrow (5,6) \rightarrow 2\}$; while v_1 and v_3 have second-order out-degree proximity, because of $\{1 \rightarrow (7,8) \leftarrow 3\}$ in the Figure 2.c.ii.

In directed graph, the edges of vertex v_k adjacent to v_i and v_j is pair-wise, thus, $A_{S_{in}}(i,j) = A_{S_{in}}(j,i)$ and $A_{S_{out}}(i,j) = A_{S_{out}}(j,i)$. In other words, $\mathbf{A}_{S_{in}}$ and $\mathbf{A}_{S_{out}}$ are symmetric.

4.2 Directed Graph Convolution

In the previous section, we define the first-order and second-order proximity, and have obtained three proximity symmetric matrices $\mathbf{A}_F, \mathbf{A}_{S_{in}}$ and $\mathbf{A}_{S_{out}}$. Similar to the authors that define the graph convolution operation on undirected graphs in Section 3, we use first- and second-order proximity matrix to achieve graph convolution of directed graphs.

In Equation 5, the adjacency matrices $\mathbf{A}'$ of the graph stores the information of the graph and provides the receptive field for the filter Θ, so as to realize the transformation from the signal $\mathbf{X}$ to the convolved signal $\mathbf{Z}'$. It is worth noting that the first- and second-order proximity matrix we have defined have similar functions: first-order proximity provides a '1-hop' like receptive field, and second-order proximity provides a '2-hop' like receptive field. Besides, we have the following theorem of first- and second-order proximity:

Theorem 1. *The the Laplacian of the first- and second-order proximity $\mathbf{A}_F, \mathbf{A}_{S_{in}}$ and $\mathbf{A}_{S_{out}}$ are positive semi-definite matrices.*

Proof 1. *According to Definition 6 and 7, $\mathbf{A}_F, \mathbf{A}_{S_{in}}$ and $\mathbf{A}_{S_{out}}$ are symmetric. We can consider these three matrices as weighted adjacency matrices $\mathbf{A}'$ with non-negative weights of weighted undirected graphs $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$, so as to formulate Laplacian matrix $\mathbf{L}'$ of $\mathcal{G}'$ in this format: $\mathbf{L}' = \sum_{uv \in E'} A'_{uv} T_{uv}$, where $e_i \in \mathbb{R}^N$ is the i^{th} standard basis vector, $t_{uv} = e_u - e_v$ and $T_{uv} = t_{uv} t_{uv}{}^T$. Since T_{uv} is positive semi-definite, $\mathbf{L}'$ is a weighted sum of non-negative coefficients with positive semi-definite matrices, which implies $\mathbf{L}'$ is positive semi-definite.* ∎

It should be noted that, the Laplacian of $\mathbf{A}'$ is also positive semi-definite. These features allows us to use these three matrices for spectral convolution.

Proximity Convolution

Based on the above analogy, we define the first-order proximity convolution $f_F(\mathbf{X}, \tilde{\mathbf{A}}_\mathbf{F})$, second-order in- and out-degree proximity convolution $f_{S_{in}}(\mathbf{X}, \tilde{\mathbf{A}}_{\mathbf{S_{in}}})$ and $f_{S_{out}}(\mathbf{X}, \tilde{\mathbf{A}}_{\mathbf{S_{out}}})$:

$$\begin{aligned} \mathbf{Z}_F &= f_F(\mathbf{X}, \tilde{\mathbf{A}}_\mathbf{F}) = \tilde{\mathbf{D}}_\mathbf{F}^{-\frac{1}{2}} \tilde{\mathbf{A}}_\mathbf{F} \tilde{\mathbf{D}}_\mathbf{F}^{-\frac{1}{2}} \mathbf{X}\Theta \\ \mathbf{Z}_{S_{in}} &= f_{S_{in}}(\mathbf{X}, \tilde{\mathbf{A}}_{\mathbf{S_{in}}}) = \tilde{\mathbf{D}}_{\mathbf{S_{in}}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{\mathbf{S_{in}}} \tilde{\mathbf{D}}_{\mathbf{S_{in}}}^{-\frac{1}{2}} \mathbf{X}\Theta \\ \mathbf{Z}_{S_{out}} &= f_{S_{out}}(\mathbf{X}, \tilde{\mathbf{A}}_{\mathbf{S_{out}}}) = \tilde{\mathbf{D}}_{\mathbf{S_{out}}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{\mathbf{S_{out}}} \tilde{\mathbf{D}}_{\mathbf{S_{out}}}^{-\frac{1}{2}} X\Theta \end{aligned}, \tag{11}$$

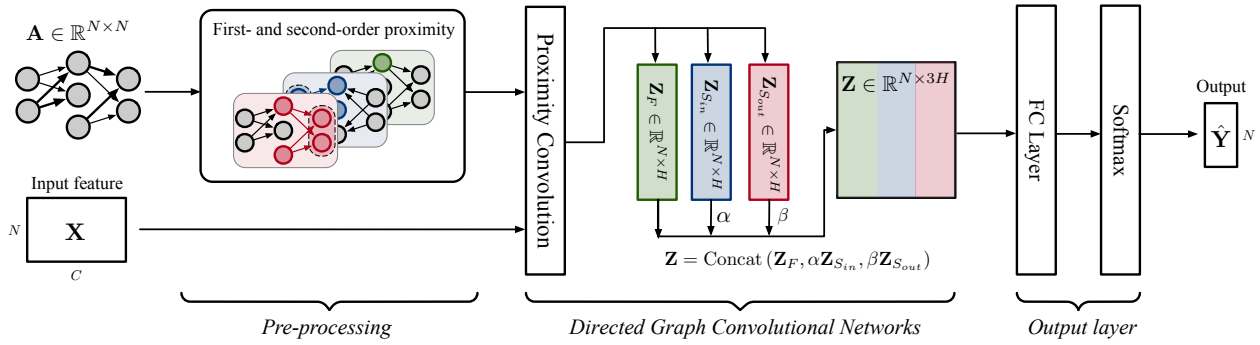


Figure 3: The schematic depiction of DGCN for semi-supervised learning. Model inputs are an adjacent matrix A and a features matrix X, while outputs are labels of predict nodes $\hat{\mathbf{Y}}$.

where adjacent weighted matrix $\tilde{\mathbf{A}}$ ($\mathbf{A}$ added self-loop) is used in the definition to derive $\tilde{\mathbf{A}}_{\mathbf{F}}$, $\tilde{\mathbf{A}}_{\mathbf{S_{in}}}$ and $\tilde{\mathbf{A}}_{\mathbf{S_{out}}}$. $\tilde{\mathbf{D}}_{\mathbf{F}} = diag(\sum_j^n \tilde{A}_F(i,j))$, $\tilde{\mathbf{D}}_{\mathbf{S_{in}}} = diag(\sum_j^n \tilde{A}_{S_{in}}(i,j))$ and $\tilde{\mathbf{D}}_{\mathbf{S_{out}}} = diag(\sum_j^n \tilde{A}_{S_{out}}(i,j))$.

It can be seen that $\mathbf{Z}_F, \mathbf{Z}_{S_{in}}$ and $\mathbf{Z}_{S_{out}}$ not only obtain rich first- and second-order neighbor feature information, but $\mathbf{Z}_{S_{in}}$ and $\mathbf{Z}_{S_{out}}$ also retain the directed graph structure information. Based on these facts, we further design a fusion method to integrate the three signals together, so as to retain the characteristics of the directed structure while obtaining the surrounding information.

Fusion Operation

Directed graph fusion operation Γ is a signal fusion function of the first-order proximity convolution output $\mathbf{Z}_F$, second-order in- and out-degree proximity convolution outputs $\mathbf{Z}_{S_{in}}$ and $\mathbf{Z}_{S_{out}}$:

$$\mathbf{Z} = \Gamma(\mathbf{Z}_F, \mathbf{Z}_{S_{in}}, \mathbf{Z}_{S_{out}}). \tag{12}$$

Fusion function Γ can be various, such as normalization functions, summation functions, and concatenation. In practice, we find concatenation fusion has the best performance. A simple example is:

$$\mathbf{Z} = Concat(\mathbf{Z}_F, \alpha\mathbf{Z}_{S_{in}}, \beta\mathbf{Z}_{S_{out}}), \tag{13}$$

where $Concat(\cdot)$ is the concatenation of matrices, α and β are weights to control the importance between different proximities. For example, in a graph with fewer second-order neighbors, we can reduce the values of α and β and use more first-order information. α and β can be set manually or trained as learnable parameters.

For a given features matrix $\mathbf{X}$ and a directed adjacency matrix $\mathbf{A}$, after taking all the steps above, we can get the final directed graph result $Z = f(\mathbf{X}, A)$.

4.3 Implementation

In the previous section, we proposed a simple and flexible model on the directed graph, which can extract the surrounding information efficiently and retain the directed structure. In this section, we will implement our model to solve semi-supervised node classification task. More specifically, how to mine the similarity between node class using weighted adjacency matrix $\mathbf{A}$ when there is no graph structure information in node feature matrix $\mathbf{X}$.

For this task, we build a two layer network model on directed graph with a weighted adjacency matrix $\mathbf{A}$ and node feature matrix $\mathbf{X}$, is schematically depicted in Figure 3. In the first step, we calculate the first- and second-order proximity matrixes $\hat{\mathbf{A}}_{\mathbf{F}}$,$\hat{\mathbf{A}}_{\mathbf{S_{in}}}$ and $\hat{\mathbf{A}}_{\mathbf{S_{out}}}$ according to Equation 7 in the preprocessing stage. Our model can be written in the following form of forward propagation:

$$\hat{\mathbf{Y}} = f(\mathbf{X}, \mathbf{A}) = softmax\left(Concat\left(ReLU\left(\begin{matrix}\hat{\mathbf{A}}_{\mathbf{F}}\,\mathbf{X}\,\Theta^{(0)} \\ \alpha\,\hat{\mathbf{A}}_{\mathbf{S_{in}}}\,\mathbf{X}\,\Theta^{(0)} \\ \beta\,\hat{\mathbf{A}}_{\mathbf{S_{out}}}\,\mathbf{X}\,\Theta^{(0)}\end{matrix}\right)\Theta^{(1)}\right)\right). \tag{14}$$

In this formula, the first layer is the directed graph convolution layer. Three different proximity convolutions share a same filter weight matrix $\Theta^{(0)} \in \mathbb{R}^{C \times H}$, which can transform the input dimension C to the embedding size H. After feature matrix $\mathbf{X}$ through the first layer, there will be three different convolved results. Then we use a fusion function to concatenate them together, α and β are variable weights to trade off first- and second-order feature embedding. The second layer is a fully connected layer, which we use to change feature dimension from $3H$ to F. $\Theta^{(1)} \in \mathbb{R}^{3H \times F}$ is an embedding-to-output weight matrix. The softmax function is defined as $\text{softmax}(x_i) = \frac{1}{\mathcal{Z}}\exp(x_i)$ with $\mathcal{Z} = \sum_i \exp(x_i)$. We use all labeled examples to evaluate the cross-entropy error for semi-supervised node classification task:

$$\mathcal{L} = -\sum_{l \in \mathbf{V}_L}\sum_{f=1}^{F} Y_{lf} \ln Z_{lf} \tag{15}$$

where Y_i is the label of vertex i and $\mathcal{V}_L$ is the subset of $\mathcal{V}$ which is labeled. The pseudocode of DGCN is attached in the Appendix.

4.4 Discussion

4.4.1 Time and Space Complexity. For graph convolution defined in the Equation 11, we can use a sparse matrix to store weighted adjacency matrix $\mathbf{A}$. Because we use full batch training in this task, full dataset has to be loaded into memory for every iteration. The memory space cost is $O(|\mathcal{E}|)$, which means it is linear with the number of edges.

At the same time, we use the sparse matrix and the density matrix to multiply during the convolution operation. The multiplication of the sparse matrix can be considered to be linearly related to the number of edges $|\mathcal{E}|$. In the semi-supervised classification task, we

need to multiply with $\Theta^0 \in \mathbb{R}^{C\times H}$ and $\Theta^1 \in \mathbb{R}^{3H\times F}$. Thus, we can obtain the computational complexity of the model as $O(|\mathcal{E}|CHF)$.

4.4.2 Generalization to other Graph Models. Our method using first-and second-order proximity to improve the convolution receptive field and retain directed information has strong generalization ability. In most spectral-based models, we can use these proximity matrices to replace the original adjacency matrix.

Take Simplifying Graph Convolutional Networks (SGC)[31] as an example, we can generalize our method to the SGC model as follows:

$$\hat{\mathrm{Y}}_{\mathrm{S}'} = \mathrm{softmax}\left(\mathrm{S}'\mathrm{X}\Theta\right), \tag{16}$$

where we use the concatenation of first- and second-order proximity matrix $\mathbf{A_F}$,$\mathbf{A_{S_{in}}}$ and $\mathbf{A_{S_{out}}}$ to replace the origin K-th power of adjacency matrix, S^K, S is the simplified adjacency matrix defined in SGC[31]. Experimental results in Section 5.3 show that integrating our method can not only make SGC model simpler, but also improve accuracy.

4.4.3 Relation with K-hop Methods. Our work considers not only the first-order relationships, but also the second-order ones when extracting surrounding information. The first-order proximity has the similar function to the 1-hop, which is to obtain the information of directly connected points. However, the reason why we do not define our second-order relationship as 2-hop is that it does not need node i and node j to have a 2 degree path directly.

For the K-hop method, and $K = 2$, they need a K degree path from i to j, i.e. $\{i \to k \to j\}$ in directed graph. However, in our method, the second-order pattern diagram is transformed into $\{i \to k \leftarrow j\}$ and $\{i \leftarrow k \to j\}$, which is obvious that we get information from the shared attributes among nodes, not from the path. What's more, when evaluating the second-order proximity of nodes, we do not use the weights of the connecting edges between them, but use the sum of the normalized weights of their shared nodes.

5 EXPERIMENTS

In this section, we evaluate the effectiveness of our model using experiments. We test on citation and co-purchase networks, and then evaluate the performance on directed and undirected dataset.

5.1 Datasets and Baselines

We use the several datasets to evaluate our model. In the citation network datasets: Cora-Full [2], Cora-ML [2], CiteSeer [23] , DBLP [19] and PubMed [18], nodes represent articles, while edges represent citation between articles. These datasets also include bag-of-words feature vectors for each article. In addition to the citation network, we also use the Amazon Co-purchase Network: Amazon-Photo and Amazon-Computers [24], where nodes represent goods, while edges represent two kinds of goods that are often purchased together. Bag-of-words encoded product reviews product category are also given as features, and class labels are given by the product category. In the above datasets, except DBLP and PubMed are undirected data we obtained, the rest are directed. The statistics of datasets are summarized in Appendix.

We compare our model to five state-of-the-art models that can be divided into two main categories: 1) **spectral-based** GNNs including ChebNet [4], Graph Convolutional Networks (GCNs) [11], Simplifying Graph Convolutional Networks (SGC) [31] and 2) **spatial-based** GNNs containing GraphSage [7] and Graph Attention Networks (GAT) [27]. The descriptions and settings of them are introduced in the Appendix.

5.2 Experimental Setup

Our Method Setup We train the two-layer DGCN model built in Section 4.3 for semi-supervised node classification task and provide additional experiments to explore the effect of model layers on the accuracy. We use full batch training, and each iteration will use the whole dataset. For each epoch, we initialize weights according to Glorot and Bengio[5] and initialize biases with zeros. We use Adam[10] as optimizer with a learning rate of 0.01. Validation set is using for hyperparameter optimization, which have weights(α, β) of first- and second-order proximity, dropout rate for all layer, L2 regularization factor for the DGCN layer and embedding size.

Dataset Split The split of the dataset will greatly affect the performance of the model[13, 24]. Especially for a single split, not only will it cause overfitting problems during training, but it is also easy to get misleading results. In our experiments, we will randomly split the data set and perform multiple experiments to obtain stable and reliable results. What's more, we also test the model under different sizes of training set in Section 5.3. For train/validation/test splitting, we choose 20 labels per class for training set, 500 labels for validation set and rest for test set, which follows the split in GCN[11], which marked as ***Label Split***.

5.3 Experimental Results

Semi-Supervised Node Classification

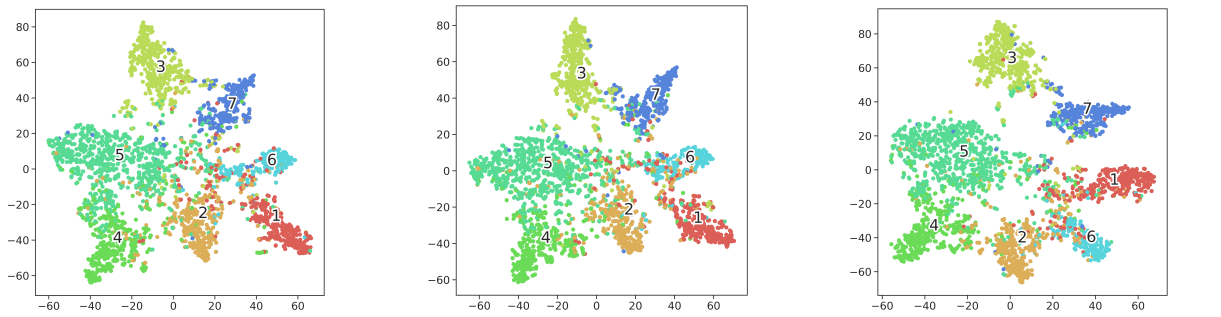
The comparison results of our model and baselines on seven datasets are reported in Table 1. Reported numbers denote classification accuracy in percent. Except DBLP and PubMed, all other datasets are directed. We train all models for a maximum of 500 epochs and early stop if the validation accuracy does not increase for 50 consecutive epochs in each dataset split, then calculate mean test accuracy and standard deviation averaged over 10 random train/validation/test splits with 5 random weight initializations. We use the following settings of hyerparameters for all datasets: drop rate is 0.5; L2 regularization is $5 \cdot 10^{-4}$; $\alpha = \beta = 1$ (we will explain the reasons in later section). Besides, we choose embedding size as 128 for Co-purchase Network: Amazon-Photo and Amazon-Computer, and 64 for others.

Our method achieved the state-of-the-art results on all datasets except Cora-Full. Although SGC achieves the best results on Cora-Full, its performances on other datasets are not outstanding. Our method achieves best results on both directed (Cora-ML, CiteSeer) and undirected (DBLP, PubMed) datasets. Our method is not significantly improved compared to GCN on the Amazon-Photo and Amazon-Computer, mainly because our model has only one convolutional layer while GCN uses two convolutional layers. The single layer network representation capability is not enough to handle large nodes graph.

First- & Second-order Proximity Evaluation

Table 1: Mean test accuracy and standard deviation in percent. Underlined bold font indicates best results.

Label Split	CORA-FULL	CORA-ML	CITESEER	DBLP	PUBMED	AMAZON-PHOTO	AMAZON-COMPUTER
ChebNet	58.0 ± 0.5	79.2 ± 1.4	59.7 ± 4.0	64.0 ± 2.8	74.6 ± 2.5	82.5 ± 2.4	72.9 ± 3.0
GCN	59.1 ± 0.7	81.7 ± 1.2	64.7 ± 2.3	71.5 ± 2.7	76.8 ± 2.2	90.4 ± 1.5	81.9 ± 1.9
SGC	**61.2 ± 0.6**	80.3 ± 1.1	61.4 ± 3.4	69.2 ± 2.8	75.8 ± 2.8	89.4 ± 1.4	80.2 ± 1.2
GraphSage	58.1 ± 0.7	80.2 ± 1.6	62.8 ± 2.1	68.1 ± 2.5	75.2 ± 3.2	89.8 ± 1.9	80.4 ± 2.5
GAT	60.8 ± 0.6	81.5 ± 1.0	63.7 ± 2.0	71.8 ± 2.6	76.5 ± 2.3	90.0 ± 1.3	81.2 ± 2.5
DGCN	60.8 ± 0.6	**82.0 ± 1.4**	**65.4 ± 2.3**	**72.5 ± 2.5**	**76.9 ± 1.9**	**90.8 ± 1.1**	**82.0 ± 1.7**



(a) GCN (b) DGCN uses only First-order Proximity (c) DGCN uses both First- and Second-order Proximity

Figure 4: 2D t-SNE[17] visualizations of the first convolutional layer feature outputs on CORA-ML dataset. The data of different classes (denote by colors) are distributed more clearly and compactly in our model feature map (c).

Table 2 reports the two smoothness values of CORA-ML, CITESEER, DBLP and PUBMED. *Features Smoothness* and *Label Smoothness* are defined in Preliminaries 2.1 to measure the quantity and quality of information that models can obtain, respectively. After adding second-order proximity, the feature smoothness of CITESEER increases from 8.719×10^{-4} to 54.720×10^{-4}, while the label smoothness increases from 0.4893 to 0.5735. This change shows that the second-order proximity is very effective on this dataset, which help increase the quantity and improve quality of information from the surrounding. The label smoothness of other datasets decrease slightly, while their feature smoothness significantly increase. In other words, the second-order proximity widens the receptive field, thus greatly increases the amount of information obtained. Besides, the second-order proximity preserves the directed structure information which helps it to filter out valid information. The t-SNE results shown in Figure 4 also prove that second-order proximity can help the model achieve better classification results.

Generalization to other Model (SGC)

In order to test the generalization ability of our model, we design an experiment according to the scheme proposed in Section 4.4.2. We use the concatenation of first- and second-order proximity matrix to replace the origin K-th power of adjacency matrix. The generalized SGC model is denoted by **SGC+DGCN**. In addition, we set the power time $K = 2$ to the origin SGC model. We follow the experimental setup described in the previous section, and the results are summarized in Table 3. Obviously, our generalized model outperforms the original model on all datasets, not only significantly improves classification accuracy, but also has more stable performance (with smaller standard deviations). Our method has good generalization ability because it has a simple structure that can be plugged into existing models easily while providing a wider receptive field by the second-order proximity matrix to improve model performance.

Table 2: Smoothness values for First- and Second-order Proximity on different datasets. 1^{st} represents first-order proximity, $1^{st}\&2^{nd}$ represents first- and second-order proximity, λ_f means Feature Smoothness and λ_l means Label Smoothness.

Smoothness	CORA-ML	CITESEER	DBLP	PUBMED
1^{st} $\lambda_f(10^{-4})$	3.759	8.719	3.579	3.135
$1^{st}\&2^{nd}$ $\lambda_f(10^{-4})$	9.789	54.720	36.810	20.480
1^{st} λ_l	0.577	0.489	0.656	0.605
$1^{st}\&2^{nd}$ λ_l	0.393	0.574	0.459	0.547

Table 3: Accuracy of origin SGC and generalized SGC. Underlined bold font indicates best results.

Label Split	CORA-ML	CITESEER	DBLP	PUBMED
SGC	80.3 ± 1.1	61.4 ± 3.4	69.2 ± 2.8	75.8 ± 2.8
SGC+DGCN	**82.3 ± 1.4**	**63.8 ± 2.0**	**71.1 ± 2.3**	**76.5 ± 2.3**

Effects of Model Depth

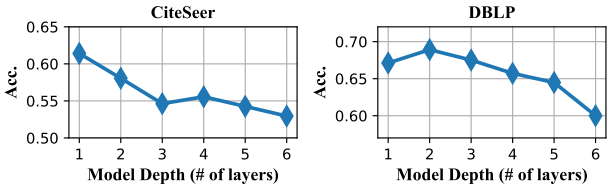


Figure 5: Effects to classification accuracy when DGCN goes deeper on CiteSeer and DBLP.

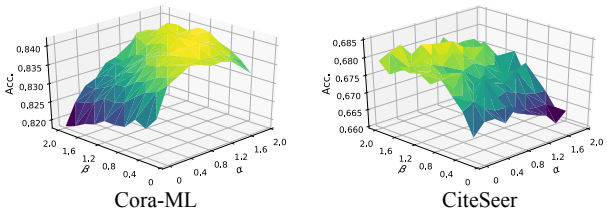
We investigate the effects of model depth (number of convolutional layers) on classification performance. To prevent overfitting with only one layer, we increase the difficulty of the task and set the training set size per class to 10, validation set size to 500 and the rest as test set. The other settings are the same with previous. Results are summarized in Figure 5. Obviously, for the datasets experimented here, the best results are obtained by a 1- or 2-layer model and test accuracy does not increase when the model goes deeper. The main reason is overfitting. The increase in the number of model layers not only greatly increases the amount of parameters, but also widens the receptive field of the convolution operation. When DGCN has only one layer, we only need to obtain information from surrounding connected nodes and nodes of shared 1-hop neighbors. When the DGCN changes to K layers, we need consider both K-hop neighborhoods and the points that share the K-hop neighbors. For a simple semi-supervised learning task, deep DGCN obtains too much information, which easily leads to overfitting.

Weights Selection of First- and Second-order Proximity

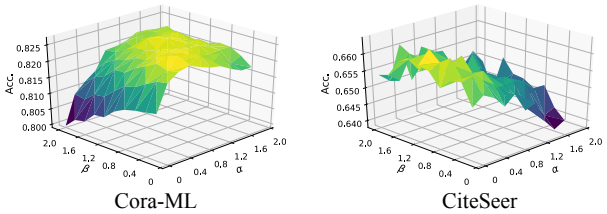
We set two hyperparameters α and β defined in Section 4.2 to adjust the first- and second-order proximity weights when concatenating them. Figure 6 shows the accuracy in validation and test set with different weights. We set α and β change within $(0, 2]$. We find that when the hyperparameters take the boundary values, the accuracies decreases significantly. When the values of the two hyperparameters are close, the accuracies of the model will increase. This is because the second-order in-degree and out-degree proximity matrix not only represent the relationship between the nodes' shared neighbors, but also encode the structure information of the graph, which needs both in- and out-degree matrix. Besides, the unbalance of second-order in- and out-degree makes the optimal hyperparameters combination differ for datasets. Therefore, we use a combination $\alpha = \beta = 1$ that can achieve balanced performance.

Effects of Training Set Size

Since the label rates for real world datasets are often small, it is important to study the performance of the model on small training set. Figure 7 shows how the number of training nodes per class to affect accuracy of different models on Cora-ML and DBLP. These four methods perform similarly under small training set size. As the amount of data increases, the accuracy greatly improves. Our method does not perform as well as GCN on CiteSeer and GAT on DBLP respectively. This can be attributed to the second-order proximity and model structure. In the case of less training data, the second-order proximity matrix will become very sparse, which makes it unable to supplement sufficient information. And our model has only one layer of convolution structure, which is not effective when we can not get enough information. On the country,



(a) Validation Accuracy



(b) Test Accuracy

Figure 6: Accuracy in validation and test set for different weights (α and β) selection on Cora-ML and CiteSeer.

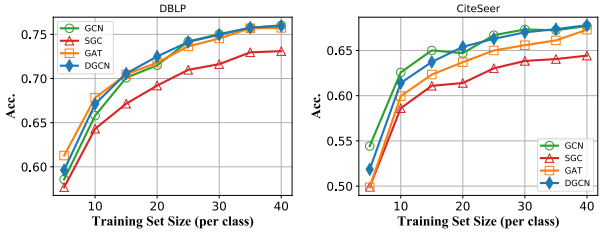


Figure 7: Accuracy for different training set sizes (number of labeled nodes per class) on DBLP and CiteSeer.

GAT uses fixed eight attention heads and GCN uses two convolutional layers to aggregate node features.

6 RELATED WORK

The field of representation learning of graph data is evolving quickly, a lot of works have contributed to it in recent years. We focus on the models similar to our method.

First- and Second-order Proximity Related Previous works have already found the powerful information extraction capabilities of second-order proximity. Zhou et al. [37] propose a regularization framework for semi-supervised classification problem, which represents directed graph in the form of bipartite graph and design smoothness function based on the *hub* and *authority* model. Tang et al.[25] propose an efficient graph embedding model for large-scale information network using first- and second-order proximity to retain graph structure information. Besides, SDNE[28] designs a semi-supervised deep model for network embedding, which combines first- and second-order proximity components to preserves highly non-linear network structure. Different from our model, when they define the first- and second-order proximity between nodes, they also consider the similarity between node feature vectors, which is not applied in our model.

K-hop Method Related Another common way to get more node information is K-hop described in many previous articles. It's used to improve the convolution receptive field. For ChebyNets[4], when set the $K = 2$, the convolution kernel can extract the information of the second-degree node from the central node. Another work is N-GCN[1], the researchers inspire from random walk, they build a multi-scale GCN which uses different powers of adjacency matrices $\mathbf{A}$ as input to achieve extract feature from different k-hop neighborhoods. It can gain the information from the k^{th} step from current node, which the same idea with K-hop. However, their K-hop methods are only applicable to undirected graphs, and our method is applicable to both types of graphs.

7 CONCLUSION AND FUTURE WORK

In this paper, we present a novel graph convolutional networks **DGCN**, which can be applied to the directed graphs. We define *first- and second-order* proximity on the directed graph to enable the spectral-based GCNs to generalize to directed graphs. It can retain directed features of graph and expand the convolution operation receptive field to extract and leverage surrounding information. Besides, we empirically show that this method helps increase the quantity and improve quality of information obtained. Finally, we use semi-supervised classification tasks and extensive experiments on various real-world datasets to validate the effectiveness and generalization capability of *first- and second-order* proximity and the improvements obtained by DGCNs over other models.

Currently, our approach is able to effectively learn directed graph representation by fused first- and second-order proximity matrices. While the selection of fusion function and concatenation weights are still manually. In the future, we will consider how to design a more principled way for fusing proximities matrices together automatically. We will also consider adapting our approach to mini-batch training in order to speed up large dataset training. In addition, we will study how to generalize our model to inductive learning.

REFERENCES

[1] Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. 2018. N-gcn: Multi-scale graph convolution for semi-supervised node classification. *arXiv preprint arXiv:1802.08888* (2018).

[2] Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815* (2017).

[3] Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. 2012. SVDFeature: a toolkit for feature-based collaborative filtering. *Journal of Machine Learning Research* 13 (2012), 3619–3622.

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

[5] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 249–256.

[6] Mark Granovetter. 1983. The strength of weak ties: A network theory revisited. *Sociological theory* (1983), 201–233.

[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.

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

[9] Yifan Hou, Jian Zhang, James Cheng, Kaili Ma, Richard T. B. Ma, Hongzhi Chen, and Ming-Chang Yang. 2020. Measuring and Improving the Use of Graph Information in Graph Neural Networks. In *ICLR*.

[10] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *Third ICLR* (2015).

[11] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[12] Jon M Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)* 46, 5 (1999), 604–632.

[13] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* (2018).

[14] Qimai Li, Xiao-Ming Wu, Han Liu, Xiaotong Zhang, and Zhichao Guan. 2019. Label efficient semi-supervised learning via graph filtering. In *CVPR*. 9582–9591.

[15] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive graph convolutional neural networks. In *AAAI*.

[16] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. 2019. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484* (2019).

[17] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[18] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. 2012. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, Vol. 8.

[19] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. 2016. Tri-party deep network representation. *Network* 11, 9 (2016), 12.

[20] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, and Charles E Leisersen. 2019. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. *arXiv preprint arXiv:1902.10191* (2019).

[21] Camille Poignard, Tiago Pereira, and Jan Philipp Pade. 2018. Spectra of Laplacian matrices of weighted graphs: structural genericity properties. *SIAM J. Appl. Math.* 78, 1 (2018), 372–394.

[22] Stephanie M Reich, Kaveri Subrahmanyam, and Guadalupe Espinoza. 2012. Friending, IMing, and hanging out face-to-face: overlap in adolescents' online and offline social networks. *Developmental psychology* 48, 2 (2012), 356.

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

[24] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[25] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.

[26] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. 2018. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735* (2018).

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[28] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1225–1234.

[29] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *SIGKDD*. 968–977.

[30] Xiaolong Wang, Yufei Ye, and Abhinav Gupta. 2018. Zero-shot recognition via semantic embeddings and knowledge graphs. In *CVPR*. 6857–6866.

[31] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. 2019. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153* (2019).

[32] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

[33] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[34] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph convolutional networks for text classification. In *AAAI*, Vol. 33. 7370–7377.

[35] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. In *NIPS*. 9240–9251.

[36] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. 2019. Graph convolutional networks: a comprehensive review. *Computational Social Networks* 6, 1 (2019), 11.

[37] Dengyong Zhou, Thomas Hofmann, and Bernhard Schölkopf. 2005. Semi-supervised learning on directed graphs. In *NIPS*. 1633–1640.

A REPRODUCIBILITY DETAILS

To support the reproducibility of the results in this paper, we details the pseudocode, the datasets and the baseline settings used in experiments. We implement the DGCN and all the baseline models using the python library of PyTorch [1] and DGL 0.3 [2]. All the experiments are conducted on a server with one GPU (NVIDIA GTX-2080Ti), two CPUs (Intel Xeon E5-2690 * 2) and Ubuntu 16.04 System.

A.1 DGCN pseudocode

Algorithm 1: DGCN procedure

Input: Graph: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;
graph adjacency matrix: $\mathbf{A}$;
features matrix: $\mathbf{X}$;
non-linear function: σ;
weight matrices: $\mathbf{\Theta}$;
concat weight: α, β
Output: Predict class matrix $\hat{\mathbf{Y}}$

Initialize $\mathbf{\Theta}$;
First- and Second-order Proximity Computation
for $i \in \mathcal{V}$ **do**
  for $j \in \mathcal{V}$ **do**
    $A_F(i,j) = A^{sym}(i,j)$
    $SUM_{in} \leftarrow 0$
    $SUM_{out} \leftarrow 0$
    for $k \in \mathcal{V}$ **do**
      if $(k,i)\&(k,j) \in \mathcal{E}$ **then**
        $SUM_{in} \leftarrow SUM_{in} + \frac{A_{k,i}A_{k,j}}{\sum_v A_{k,v}}$
      end
      if $(i,k)\&(j,k) \in \mathcal{E}$ **then**
        $SUM_{out} \leftarrow SUM_{out} + \frac{A_{i,k}A_{j,k}}{\sum_v A_{v,k}}$
      end
    end
    $A_{S_{in}}(i,j) \leftarrow SUM_{in}$
    $A_{S_{out}}(i,j) \leftarrow SUM_{out}$
  end
end
Directed Graph Convolution Networks
$\mathbf{D_F} \leftarrow \text{RowNorm}(A_F)$
$\mathbf{D_{S_{in}}} \leftarrow \text{RowNorm}(A_{S_{in}})$
$\mathbf{D_{S_{out}}} \leftarrow \text{RowNorm}(A_{S_{out}})$
$\mathbf{Z}_F \leftarrow \tilde{\mathbf{D}}_{\mathbf{F}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{\mathbf{F}} \tilde{\mathbf{D}}_{\mathbf{F}}^{-\frac{1}{2}} \mathbf{X}\Theta$
$\mathbf{Z}_{S_{in}} \leftarrow \tilde{\mathbf{D}}_{\mathbf{S_{in}}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{\mathbf{S_{in}}} \tilde{\mathbf{D}}_{\mathbf{S_{in}}}^{-\frac{1}{2}} \mathbf{X}\Theta$
$\mathbf{Z}_{S_{out}} \leftarrow \tilde{\mathbf{D}}_{\mathbf{S_{out}}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{\mathbf{S_{out}}} \tilde{\mathbf{D}}_{\mathbf{S_{out}}}^{-\frac{1}{2}} X\Theta$
$\mathbf{Z} = \sigma(\text{Concat}(\mathbf{Z}_F, \alpha\mathbf{Z}_{S_{in}}, \beta\mathbf{Z}_{S_{out}}))$
$\hat{\mathbf{Y}} = softmax(FCLayer(\mathbf{Z}))$

[1] https://pytorch.org
[2] https://www.dgl.ai

A.2 Datasets Details

We use seven open access datasets to test our method. The origin Cora-ML has 70 classes, we combine the 2 classes that cannot perform the dataset split to the nearest class.

Table 4: Datasets Details

Datasets	Nodes	Edges	Classes	Features	Label rate
Cora-Full	19793	65311	68	8710	7.07%
Cora-ML	2995	8416	7	2879	4.67%
CiteSeer	3312	4715	6	3703	3.62%
DBLP	17716	105734	4	1639	0.45%
PubMed	18230	79612	3	500	0.33%
Amazon-Photo	7650	143663	8	745	2.10%
-Computer	13752	287209	10	767	1.45%

Label rate is the fraction of nodes in the training set per class. We use 20 labeled nodes per class to calculate the label rate.

A.3 Baselines Details and Settings

The baseline methods are given below:

• **ChebNet**: It redefines graph convolution using Chebyshev polynomials to remove the time-consuming Laplacian eigenvalue decomposition.

• **GCN**: It has multi-layers which stacks first-order Chebyshev polynomials as graph convolution layer and learns graph representations use a nonlinear activation function.

• **SGC**: It removes nonlinear layers and collapse weight matrices to reduce computational consumption.

• **GraphSage**: It proposes a general inductive framework that can efficiently generate node embeddings for previously unseen data

• **GAT**: It applies attention mechanism to assign different weights to different neighborhood nodes based on node features.

For all baseline models, we use their model structure in the original papers, which including layer number, activation function selection, normalization and regularization selection, etc. It is worth noting that GraphSage has three variants in the original article using different aggregators: **mean**, **meanpool** and **maxpool**. In this paper, we use **mean** as its aggregator as it performs best [24]. Besides, we set the mini-batch size to 512 on Amazon-Photo and Amazon-Computer and 16 on other datasets. For GCN, we set the size of hidden layer to 64 on Amazon-Photo and Amazon-Computer and 16 on other datasets. We also fix the number of attention heads to 8 for GAT, power number to 2 for SGC and $k = 2$ in ChebNets, as proposed in the respective papers.