Towards Deeper Graph Neural Networks

Meng Liu
Texas A&M University
College Station, TX
mengliu@tamu.edu

Hongyang Gao
Texas A&M University
College Station, TX
hongyang.gao@tamu.edu

Shuiwang Ji
Texas A&M University
College Station, TX
sji@tamu.edu

ABSTRACT

Graph neural networks have shown significant success in the field of graph representation learning. Graph convolutions perform neighborhood aggregation and represent one of the most important graph operations. Nevertheless, one layer of these neighborhood aggregation methods only consider immediate neighbors, and the performance decreases when going deeper to enable larger receptive fields. Several recent studies attribute this performance deterioration to the over-smoothing issue, which states that repeated propagation makes node representations of different classes indistinguishable. In this work, we study this observation systematically and develop new insights towards deeper graph neural networks. First, we provide a systematical analysis on this issue and argue that the key factor compromising the performance significantly is the entanglement of representation transformation and propagation in current graph convolution operations. After decoupling these two operations, deeper graph neural networks can be used to learn graph node representations from larger receptive fields. We further provide a theoretical analysis of the above observation when building very deep models, which can serve as a rigorous and gentle description of the over-smoothing issue. Based on our theoretical and empirical analysis, we propose Deep Adaptive Graph Neural Network (DAGNN) to adaptively incorporate information from large receptive fields. A set of experiments on citation, co-authorship and co-purchase datasets have confirmed our analysis and insights and demonstrated the superiority of our proposed methods.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Artificial intelligence**; • **Mathematics of computing** → **Graph algorithms**.

KEYWORDS

deep learning, graph representation learning, graph neural networks

ACM Reference Format:
Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards Deeper Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining USB Stick (KDD '20), August 23–27, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403076

KDD '20, August 23–27, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403076

1 INTRODUCTION

Graphs, representing entities and their relationships, are ubiquitous in the real world, such as social networks, point clouds, traffic networks, knowledge graphs, and molecular structures. Recently, many studies focus on developing deep learning approaches for graph data, leading to rapid development in the field of graph neural networks. Great successes have been achieved for many applications, such as node classification [6, 8, 10, 11, 20, 29, 30, 32], graph classification [5, 7, 13, 17, 31, 33, 34, 37] and link prediction [35, 36]. Graph convolutions adopt a neighborhood aggregation (or message passing) scheme to learn node representations by considering the node features and graph topology information together, among which the most representative method is Graph Convolutional Networks (GCNs) [10]. GCN learns representation for a node by aggregating representations of its neighbors iteratively. However, a common challenge faced by GCN and most other graph convolutions is that one layer of graph convolutions only consider immediate neighbors and the performance degrades greatly when we apply multiple layers to leverage large receptive fields. Several recent works attribute this performance degradation to the over-smoothing issue [2, 14, 32], which states that representations from different classes become inseparable due to repeated propagation. In this work, we study this performance deterioration systematically and develop new insights towards deeper graph neural networks.

We first systematically analyze the performance degradation when stacking multiple GCN layers by using our quantitative metric for node representation smoothness measurement and a data visualization technique. We observe and argue that the main factor compromising the performance greatly is the entanglement of representation transformation and propagation. After decoupling these two operations, it is demonstrated that deeper graph neural networks can be deployed to learn graph node representations from larger receptive fields without suffering from performance deterioration. The over-smoothing issue is shown to affect performance only when extremely large receptive fields are utilized. We further give a theoretical analysis of the above observation when building very deep models, which shows that graph node representations will become indistinguishable when depth goes infinity. This aligns with the over-smoothing issue. The previous descriptions of the over-smoothing issue simplify the assumption of non-linear activation function [14, 32] or make approximations of different probabilities [32]. Our theoretical analysis can serve as a more rigorous and gentle description of the over-smoothing issue. Based on our theoretical and empirical analysis, we propose an efficient and effective network, termed as Deep Adaptive Graph Neural

Network, to learn node representations by adaptively incorporating information from large receptive fields. Extensive experiments on citation, co-authorship, and co-purchase datasets demonstrate the reasonability of our insights and the superiority of our proposed network.

2 BACKGROUND AND RELATED WORKS

First, we introduce our notations used throughout this paper. Generally, we let bold uppercase letters represent matrices and bold lowercase letters denote vectors. A graph is formally defined as $G = (V, E)$, where V is the set of nodes (vertices) that are indexed from 1 to n and $E \subseteq V \times V$ is the set of edges between nodes in V. $n = |V|$ and $m = |E|$ are the numbers of nodes and edges, respectively. In this paper, we consider unweighted and undirected graphs. The topology information of the whole graph is described by the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, where $\boldsymbol{A}_{(i,j)} = 1$ if an edge exists between node i and node j, otherwise 0. The diagonal matrix of node degrees are denoted as $\boldsymbol{D} \in \mathbb{R}^{n \times n}$, where $\boldsymbol{D}_{(i,i)} = \sum_j \boldsymbol{A}_{(i,j)}$. $\mathcal{N}_i$ denotes the neighboring nodes set of node i. An attributed graph has a node feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, where each row $\boldsymbol{x}_i \in \mathbb{R}^d$ represents the feature vector of node i and d is the dimension of node features.

2.1 Graph Convolution Operations

Most popular graph convolution operations follow a neighborhood aggregation (or message passing) fashion to learn a node representation by propagating representations of its neighbors and applying transformation after that. The ℓ-th layer of a general graph convolution can be described as

$$\begin{aligned} \boldsymbol{a}_i^{(\ell)} &= \text{PROPAGATION}^{(\ell)}\left(\left\{\boldsymbol{x}_i^{(\ell-1)}, \{\boldsymbol{x}_j^{(\ell-1)} | j \in \mathcal{N}_i\}\right\}\right) \\ \boldsymbol{x}_i^{(\ell)} &= \text{TRANSFORMATION}^{(\ell)}\left(\boldsymbol{a}_i^{(\ell)}\right). \end{aligned} \tag{1}$$

$\boldsymbol{x}_i^{(\ell)}$ is the representation of node i at l-th layer and $\boldsymbol{x}_i^{(0)}$ is initialized as node feature $\boldsymbol{x}_i$. Most graph convolutions, like GCN [10], GraphSAGE [8], GAT [29], and GIN [31], can be obtained under this framework by deploying different propagation and transformation mechanisms.

Without losing generalization, we focus on the Graph Convolutional Network (GCN) [10], the most representative graph convolution operation, in the following analysis. The ℓ-th layer forward-propagation process is formulated as

$$\boldsymbol{X}^{(\ell)} = \sigma\left(\widehat{\boldsymbol{A}}\boldsymbol{X}^{(\ell-1)}\boldsymbol{W}^{(\ell)}\right), \tag{2}$$

where $\boldsymbol{X}^{(\ell)} \in \mathbb{R}^{n \times d^{(\ell)}}$ and $\boldsymbol{X}^{(\ell-1)} \in \mathbb{R}^{n \times d^{(\ell-1)}}$ are the output and input node representation matrices of layer ℓ. $\widehat{\boldsymbol{A}} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$, where $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the adjacency matrix with added self-connections. $\widetilde{\boldsymbol{D}}_{(i,i)} = \sum_j \widetilde{\boldsymbol{A}}_{(i,j)}$ is the diagonal node degree matrix. $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{d^{(\ell-1)} \times d^{(\ell)}}$ is a layer-specific trainable weight matrix. σ is a non-linear activation function like ReLU [21]. Intuitively, GCN learns representation for each node by propagating neighbors' representations and conducting non-linear transformation after that. GCN is originally applied for semi-supervised classification, where only partial nodes have training labels in a graph. Thanks to the propagation process, representation of a labeled node carries the information from its neighbors that are usually unlabeled, thus training signals can be propagated to the unlabeled nodes.

2.2 Related Works

From Eq.(2), one layer GCN only considers immediate neighbors, i.e. one-hop neighborhood. Multiple layers should be applied if multi-hop neighborhood is needed. In practice, however, the performance of GCN degrades greatly when multiple layers are stacked. Several works reveal that stacking many layers can bring the over-smoothing issue, which means that representations of nodes converge to indistinguishable limits. To our knowledge, [14] is the first attempt to demystify the over-smoothing issue in the GCN model. The authors first demonstrate that the propagation process of the GCN model is a special symmetric form of Laplacian smoothing [28], which makes the representations of nodes in the same class similar, thus significantly easing the classification task. Then they show that repeatedly stacking many layers may make representations of nodes from different classes indistinguishable. The same problem is studied in [32] by analyzing the connection of nodes' influence distribution and random walk [16]. Recently, SGC [30] is proposed by reducing unnecessary complexity in GCN. The authors show that SGC corresponds to a low-pass-type filter on the spectral domain, thus deriving smoothing features across a graph. Another recent work [2] verify that smoothing is the nature of most typical graph convolutions. It is showed that reasonable smoothing makes graph convolutions work and over-smoothing results in poor performance.

Due to the potential concern of the over-smoothing issue, a limited neighborhood is usually used in practice and it is difficult to extend. However, long-range dependencies should be taken into consideration, especially for peripheral nodes. Also, small receptive fields are not enough to propagate training signals to the whole graph when the number of training nodes is limited under a semi-supervised learning setting. [14] applies co-training and self-training to overcome the limitation of shallow architectures. A smoothness regularizer term and adaptive edge optimization are proposed in [2] to relieve the over-smoothing problem. Jumping Knowledge Network [32] deploys a layer-aggregation mechanism to adaptively select a node's sub-graph features at different ranges rather than to capture equally smoothed representations for all nodes. [11] utilizes the relationship between GCN and PageRank [22] to develop a propagation mechanism based on personalize PageRank, which can preserve the node's local information while gather information from a large neighborhood. Recently, Geom-GCN [24] and non-local GNNs [15] are proposed to capture long-range dependencies for disassortative graph by designing non-local aggregators.

3 EMPIRICAL AND THEORETICAL ANALYSIS OF DEEP GNNS

In this section, we first propose a quantitative metric to measure the smoothness of graph node representations. Then we utilize this metric, along with a data visualization technique, to rethink the performance degradation when utilizing GCN layer to build

Figure 1: t-SNE visualization of node representations derived by different numbers of GCN layers on Cora. Colors represent node classes.

deep graph neural networks. We observe and argue that the entanglement of representation transformation and propagation is a prominent factor that compromises the network performance. After decoupling these two operations, deeper graph neural networks can be built to learn graph node representations from large receptive fields without suffering from performance degradation. The over-smoothing issue is shown to influence the performance only when extremely large receptive fields are adopted. Further, we provide a theoretical analysis of the above observation when building very deep models, which aligns with the conclusion of over-smoothing issue and can serve as a rigorous description of the over-smoothing issue.

3.1 Quantitative Metric for Smoothness

Smoothness is a metric that reflects the similarity of node representations. Here, we first define a similarity metric between the representations of node i and node j with their Euclidean distance:

$$D(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{2}\left\| \frac{\boldsymbol{x}_i}{\|\boldsymbol{x}_i\|} - \frac{\boldsymbol{x}_j}{\|\boldsymbol{x}_j\|} \right\|, \tag{3}$$

where $\boldsymbol{x}_i$ is the feature representation of node i and $\|\cdot\|$ denotes the Euclidean norm. The Euclidean distance is a simple but effective way to measure the similarity of two representations, especially in high dimensional space. Smaller Euclidean distance value incidates higher similarity of two representations. To remove the influence of the magnitude of feature representations, we use normalized node representations to compute their Euclidean distance, thus constraining $D(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in the range of $[0, 1]$.

Based on the similarity metric in Eq.(3), we further propose a smoothness metric SMV_i for node i, which is computed as the average distance between node i to other nodes:

$$SMV_i = \frac{1}{n-1} \sum_{j \in V, j \neq i} D(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{4}$$

Hence, SMV_i measures the similarity of node i's representations to the entire graph. For instance, a node in the periphery or a leaf node usually has a large smoothness metric value. Further, we can use SMV_G to represent the smoothness metric value of the whole graph G. Its mathematical expression is defined as:

$$SMV_G = \frac{1}{n} \sum_{i \in V} SMV_i. \tag{5}$$

Here, SMV_G is negatively related to the overall smoothness of nodes' representations in graph G.

3.2 Why Deeper GNNs Fail?

In this section, we utilize our proposed smoothness metric to investigate the performance deterioration phenomenon in deep graph neural networks. Here, we mainly use the GCN layer for analysis, but the main results can be easily applied to other graph deep learning methods. Besides using our proposed metric from the quantitative perspective, we employ a data visualization technique t-SNE [18]. t-SNE provides an interpretable visualization, especially on high-dimensional data. t-SNE is capable of capturing both the local structure and the global structure like clusters in high-dimensional data, which is consistent with the classification of nodes. Hence, we utilize t-SNE to demonstrate the discriminative power of node representations in the graph.

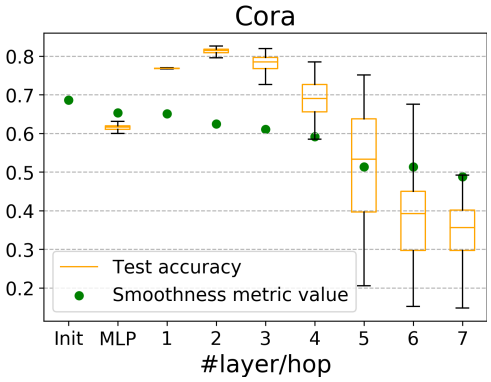

Figure 2: Test accuracy and smoothness metric value of node representations with different numbers of GCN layers on Cora. "Init" means the smoothness metric value of the original data.

We develop a series of graph neural networks (GNNs) with different depths in terms of the number of GCN layers, and evaluate them on three citation datasets; those are Cora, CiteSeer and PubMed [25]. In particular, we also include a graph neural network with depth of 0, which is approximated with a multi-layer perceptron network. A GNN with depth of 0 can be viewed as aggregating information from a 0-hop neighborhood, which only considers node features while with the graph structure ignored. We conduct 100 runs for each model on each dataset, using the same data split scheme as [10].

The result on Cora is illustrated in Figure 2. We provide results on other datasets in Section A.1 in the appendix. We can observe that test accuracy increases as the rise of the number of layers in the beginning, but degrades dramatically from 3 layers. Besides, from the t-SNE visualization on Cora in Figure 1 (t-SNE visualization results of other datasets are provided in Appendix A.3.), the discriminative power of the node representations derived by different numbers of GCN layers has the similar trend. The node representations generated by multiple GCN layers, like 6 layers, are very difficult to be separated.

Figure 3: t-SNE visualization of node representations derived by models as Eq.(6) with different numbers of layers on Cora. Colors represent node classes.

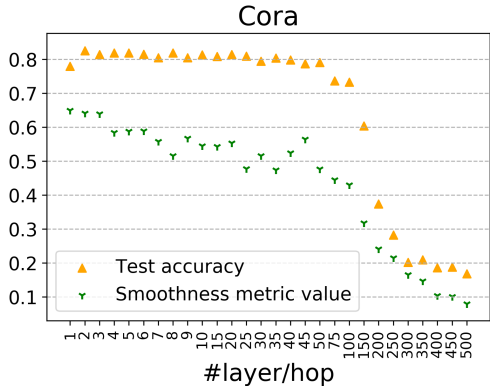

Figure 4: Test accuracy and smoothness metric value of node representations with different numbers of layers adopted in models as Eq.(6) on Cora.

Several studies [2, 14] attribute this performance degradation phenomenon to the over-smoothing issue. However, we question this view for the following two reasons. First, we hold that the over-smoothing issue only happens when node representations propagate repeatedly for a large number of iterations, especially for a graph with sparsely connected edges. As shown in Table 1, all these three citation datasets have a small value of edge density, hence several propagation iterations are not enough to make over-smoothing happen, theoretically. Second, as shown in Figure 2, the smoothness metric value of graph node representations has a slight downward trend as the number of propagation iterations increases. According to [14], the node representations suffering from the over-smoothing issue will converge to the same value or be proportional to the square root of the node degree, where the corresponding smoothness metric value should be close to 0, computed by our quantitative metric. However, the metric value in our experiments is relatively far from the ideal over-smoothing situation.

In this work, we argue that it is the entanglement of transformation and propagation that significantly compromise the performance of deep graph neural networks. Our argument is originated from the following two intuitions. First, the entanglement of representation transformation and propagation makes the number of parameters in transformation intertwined with the receptive fields in propagation. As illustrated in Eq.(1), one hop propagation requires a transformation function, thus leading to a large number of parameters when considering a large receptive field. Hence, it might be hard to train a deep GNN with a large number of parameters. This can possibly explain why the performance of multiple GCN layers in Figure 2 fluctuates greatly. Second, representation propagation and transformation should be viewed as two separate operations. Note that the class of a node can be totally predictable by its initial features, which explains why MLP, as shown in Figure 2 and 1, performs well without using any graph structure information. Propagation based on the graph structure can help to ease the classification task by making node representations in the same class to be similar, under the assumption that connected nodes usually belong to the same class. For instance, intuitively, the class of a document is completely determined by its content (i.e. its feature derived by word embedding), instead of the references relationships with other documents. Utilizing its neighbors' features just eases the classification of documents. Hence, representation transformation and propagation play their distinct roles from feature and structure aspects, respectively.

To support and verify our argument, we decouple the propagation and transformation in Eq.(2), leading to the following model:

$$\begin{aligned} \boldsymbol{Z} &= \mathrm{MLP}\left(\boldsymbol{X}\right) \\ \boldsymbol{X}_{out} &= \mathrm{softmax}\left(\widehat{\boldsymbol{A}}^{k}\boldsymbol{Z}\right). \end{aligned} \tag{6}$$

$\boldsymbol{Z} \in \mathbb{R}^{n\times c}$ denotes the new feature matrix transformed from the original feature matrix by an MLP network, where c is the number of classes. After the transformation, we apply a k-steps propagation to derive the output feature matrix $\boldsymbol{X}_{out} \in \mathbb{R}^{n\times c}$. A softmax classifier is applied to compute the classification probabilities. Notably, the separation of transformation and propagation processes is also adopted in [11] and [30] but for the sake of reducing complexity. In this work, we analyze this scheme systematically and reveal that it can help to build deeper models without suffering from performance degradation, which has not been prioritized by the community.

The test accuracy and smoothness metric value of representations with different numbers of layers adopted in Eq.(6) on Cora are illustrated in Figure 4 (Results for other datasets are shown in Appendix A.2). After resolving the entanglement of feature transformation and propagation, deeper models is capable of leveraging larger receptive fields without suffering from performance degradation. We can observe that the over-smoothing issue starts compromising the performance at an extremely large receptive field, such as 75-hop on Cora. The smoothness metric value decreases greatly after that, which is demonstrated by the metric value of nearly 0. Besides, from the t-SNE visualization in Figure 3 (The t-SNE visualization results of other datasets are provided in Appendix A.4), deep models with large receptive fields, like 50-hop, still generating distinguishable node representations, which is impressive compared to the regular GCN model. In practice, we usually do not need an extremely large receptive field because the highest shortest path distance in a connected component usually is an acceptable small

number. Thus training signals can be propagated to the entire graph with a small number of layers. This is demonstrated by the fact that graph neural networks with 2 or 3 GCN layers usually perform competitively. However, deep models with large receptive fields are necessary to incorporate more information, especially with limited training nodes under a semi-supervised learning setting.

3.3 Theoretical Analysis of Very Deep Models

The empirical analysis in the previous section shows that decoupling transformation and propagation can help to build much deeper models which can leverage larger receptive fields to incorporate more information. In this section, we provide a theoretical analysis of the above observation when building very deep graph neural networks, which aligns with the over-smoothing issue. [14] and [32] study the over-smoothing issue from the perspective of Laplacian smoothing and nodes' influence distribution, with several simplified assumptions like non-linear transformation and probability approximations. After decoupling transformation from propagation, our theoretical analysis can serve as a more rigorous and gentle description of the over-smoothing issue. In this section, we strictly describe the over-smoothing issue for 2 typical propagation mechanisms.

$\widehat{\boldsymbol{A}}_{\oplus} = \widetilde{\boldsymbol{D}}^{-1}\widetilde{\boldsymbol{A}}$ and $\widehat{\boldsymbol{A}}_{\odot} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$, where $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$, are two frequently utilized propagation mechanisms. The row-averaging normalization $\widehat{\boldsymbol{A}}_{\oplus}$ is adopted in GraphSAGE [8] and DGCNN [37]. The symmetrical normalization scheme $\widehat{\boldsymbol{A}}_{\odot}$ is applied in GCN [10]. In the following, we describe the over-smoothing issue by proving the convergence of $\widehat{\boldsymbol{A}}^k_{\oplus}$ and $\widehat{\boldsymbol{A}}^k_{\odot}$, respectively, when k goes to infinity.

Let $\boldsymbol{e} = [1, 1, \cdots, 1] \in \mathbb{R}^{1\times n}$ be a row vector whose all entries are 1. Function $\Psi(\boldsymbol{x}) = \frac{\boldsymbol{x}}{\text{sum}(\boldsymbol{x})}$ normalizes a vector to sum to 1 and function $\Phi(\boldsymbol{x}) = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$ normalizes a vector such that its magnitude is 1.

Theorem 3.1. *Given a connected graph G, $\lim_{k\to\infty} \widehat{\boldsymbol{A}}^k_{\oplus} = \boldsymbol{\Pi}_{\oplus}$, where $\boldsymbol{\Pi}_{\oplus}$ is the matrix with all rows are $\boldsymbol{\pi}_{\oplus}$ and $\boldsymbol{\pi}_{\oplus} = \Psi(\boldsymbol{e}\widetilde{\boldsymbol{D}})$.*

Theorem 3.2. *Given a connected graph G, $\lim_{k\to\infty} \widehat{\boldsymbol{A}}^k_{\odot} = \boldsymbol{\Pi}_{\odot}$, where $\boldsymbol{\Pi}_{\odot} = \Phi(\widetilde{\boldsymbol{D}}^{\frac{1}{2}}\boldsymbol{e}^T)(\Phi(\widetilde{\boldsymbol{D}}^{\frac{1}{2}}\boldsymbol{e}^T))^T$.*

From the above two theorems, we can derive the exact convergence value of $\widehat{\boldsymbol{A}}^k_{\oplus}$ and $\widehat{\boldsymbol{A}}^k_{\odot}$, respectively, when k goes to infinity in an infinite deep model. Hence, applying infinite layers to propagate information iteratively is equivalent to utilizing $\boldsymbol{\Pi}_{\oplus}$ or $\boldsymbol{\Pi}_{\odot}$ to propagate features by one step. Rows of $\boldsymbol{\Pi}_{\oplus}$ are the same and rows of $\boldsymbol{\Pi}_{\odot}$ are proportional to the square root value of the corresponding nodes' degrees. Therefore, rows of $\boldsymbol{\Pi}_{\oplus}$ or $\boldsymbol{\Pi}_{\odot}$ are linearly inseparable and utilizing them as propagation mechanism will generate indistinguishable representations, thereby leading to the over-smoothing issue.

To prove these two theorems, we first introduce the following two lemmas. The proofs of these two lemmas can be found in Appendix A.5 and A.6.

Lemma 3.3. *Given a graph G, λ is an eigenvalue of $\widehat{\boldsymbol{A}}_{\oplus}$ with left eigenvector $\boldsymbol{v}_l \in \mathbb{R}^{1\times n}$ and right eigenvector $\boldsymbol{v}_r \in \mathbb{R}^{n\times 1}$ if and only if λ is an eigenvalue of $\widehat{\boldsymbol{A}}_{\odot}$ with left eigenvector $\boldsymbol{v}_l\widetilde{\boldsymbol{D}}^{-\frac{1}{2}} \in \mathbb{R}^{1\times n}$ and right eigenvector $\widetilde{\boldsymbol{D}}^{\frac{1}{2}}\boldsymbol{v}_r \in \mathbb{R}^{n\times 1}$.*

Lemma 3.4. *Given a connected graph G, $\widehat{\boldsymbol{A}}_{\oplus}$ and $\widehat{\boldsymbol{A}}_{\odot}$ always have an eigenvalue 1 with unique associated eigenvectors and all other eigenvalues λ satisfy $|\lambda| < 1$. The left and right eigenvectors of $\widehat{\boldsymbol{A}}_{\oplus}$ associated with eigenvalue 1 are $\boldsymbol{e}\widetilde{\boldsymbol{D}} \in \mathbb{R}^{1\times n}$ and $\boldsymbol{e}^T \in \mathbb{R}^{n\times 1}$, respectively. For $\widehat{\boldsymbol{A}}_{\odot}$, they are $\boldsymbol{e}\widetilde{\boldsymbol{D}}^{\frac{1}{2}} \in \mathbb{R}^{1\times n}$ and $\widetilde{\boldsymbol{D}}^{\frac{1}{2}}\boldsymbol{e}^T \in \mathbb{R}^{n\times 1}$.*

Proof. (of Theorem 3.1) $\widehat{\boldsymbol{A}}_{\oplus}$ can be viewed as a transition matrix because all entries are nonnegative and each row sums to 1. The graph G can be further regarded as a Markov chain, whose transition matrix $\boldsymbol{P}$ is $\widehat{\boldsymbol{A}}_{\oplus}$. This Markov chain is irreducible and aperiodic because the graph G is connected and self-loops are included in the connectivity. If a Markov chain is irreducible and aperiodic, then $\lim_{k\to\infty} \boldsymbol{P}^k = \boldsymbol{\Pi}$, where $\boldsymbol{\Pi}$ is the matrix with all rows equal to $\boldsymbol{\pi}$ and $\boldsymbol{\pi}$ can be computed by $\boldsymbol{\pi}\boldsymbol{P} = \boldsymbol{\pi}$, s.t. $\sum_i \boldsymbol{\pi}_i = 1$[12]. It is obvious that $\boldsymbol{\pi}$ is the unique left eigenvector of $\boldsymbol{P}$ and is normalized such that all entries sum to 1. Hence, $\lim_{k\to\infty} \widehat{\boldsymbol{A}}^k_{\oplus} = \boldsymbol{\Pi}_{\oplus}$, where $\boldsymbol{\Pi}_{\oplus}$ is the matrix with all rows are $\boldsymbol{\pi}_{\oplus}$ and $\boldsymbol{\pi}_{\oplus} = \Psi(\boldsymbol{e}\widetilde{\boldsymbol{D}})$ from Lemma 3.4. □

Proof. (of Theorem 3.2) Although $\widehat{\boldsymbol{A}}_{\odot}$ cannot be processed as a transition matrix like $\widehat{\boldsymbol{A}}_{\oplus}$, it is a symmetric matrix, which is diagonalizable. We have $\widehat{\boldsymbol{A}}_{\odot} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$, where $\boldsymbol{Q}$ is an orthogonal matrix whose columns are normalized eigenvectors of $\widehat{\boldsymbol{A}}_{\odot}$ and $\boldsymbol{\Lambda}$ is the diagonal matrix whose diagonal entries are the eigenvalues. Then the k-th power of $\widehat{\boldsymbol{A}}_{\odot}$ can be computed by

$$\widehat{\boldsymbol{A}}^k_{\odot} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T \cdots \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T = \boldsymbol{Q}\boldsymbol{\Lambda}^k\boldsymbol{Q}^T = \sum_{i=1}^{k} \lambda_i^n \boldsymbol{v}_i \boldsymbol{v}_i^T, \tag{7}$$

where $\boldsymbol{v}_i$ is the normalized right eigenvector associated with λ_i. From Lemma 3.4, $\widehat{\boldsymbol{A}}_{\odot}$ always has an eigenvalue 1 with unique associated eigenvectors and all other eigenvalues λ satisfy $|\lambda| < 1$. Hence, $\lim_{k\to\infty} \widehat{\boldsymbol{A}}^k_{\odot} = \Phi(\widetilde{\boldsymbol{D}}^{\frac{1}{2}}\boldsymbol{e}^T)(\Phi(\widetilde{\boldsymbol{D}}^{\frac{1}{2}}\boldsymbol{e}^T))^T$. □

These two theorems hold for connected graphs that are frequently studied in graph neural networks. For a disconnected graph, these theorems can also be applied to each of its connected components, which means that applying these propagation mechanisms infinite times will generate indistinguishable node representations in each connected components.

The above theorems reveal that over-smoothing will make node representations inseparable and provide the exact convergence value of frequently used propagation mechanisms. Theoretically, we have proved that the over-smoothing issue is inevitable in very deep models. Further, the convergence speed is a more important factor that we should consider in practice. Mathematically, according to Eq.(7), the convergence speed depends on the other eigenvalues except 1 of the propagation matrix, especially the second largest eigenvalue. Intuitively, the propagation matrix is determined by the topology information of the corresponding graph. This might be the reason for our observation in Section 3.2 that a sparsely connected graph suffers from the over-smoothing only when extremely deep models are applied.

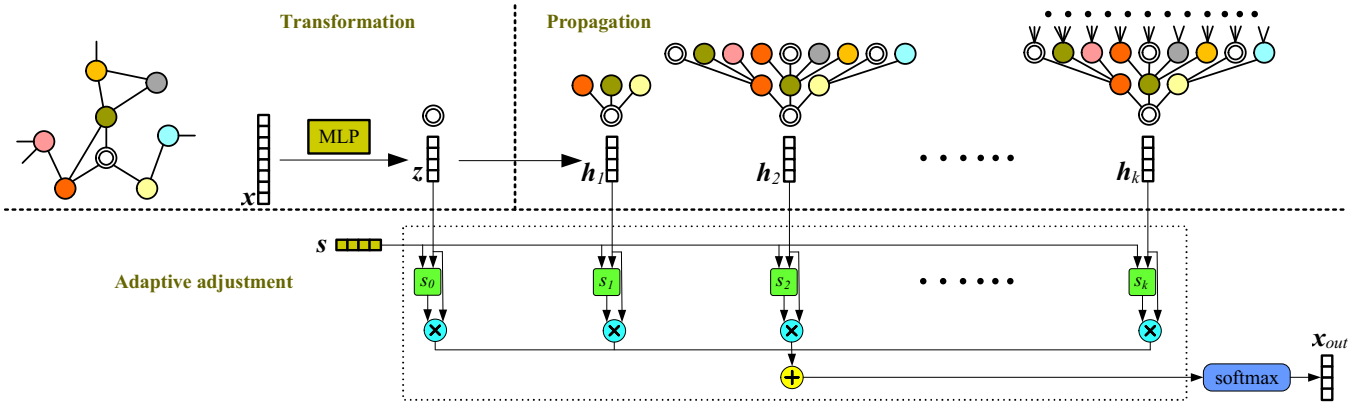

Figure 5: An illustration of the proposed Deep Adaptive Graph Neural Network (DAGNN) . For clarity, we show the pipeline to generate the prediction for one node. Notation letters are consistent with Eq.(8) but bold lowercase versions are applied to denote representation vectors. s is the projection vector that computes retainment scores for representations generating from various receptive fields. s_0, s_1, s_2, and s_k represent the retainment scores of z, h_1, h_2, and h_k, respectively.

4 DEEP ADAPTIVE GRAPH NEURAL NETWORK

In this section, we propose Deep Adaptive Graph Neural Network (DAGNN) based on the above insights. Our DAGNN contributes two prominent advantages. First, it decouples the representation transformation from propagation so that large receptive fields can be applied without suffering from performance degradation, which has been verified in Section 3.2. Second, it utilizes an adaptive adjustment mechanism that can adaptively balance the information from local and global neighborhoods for each node, thus leading to more discriminative node representations. The mathematical expression of DAGNN is defined as

$$\begin{aligned}
\boldsymbol{Z} &= \mathrm{MLP}(\boldsymbol{X}) && \in \mathbb{R}^{n\times c}\\
\boldsymbol{H}_\ell &= \widehat{\boldsymbol{A}}^\ell \boldsymbol{Z}, \ell = 1, 2, \cdots, k && \in \mathbb{R}^{n\times c}\\
\boldsymbol{H} &= \mathrm{stack}(\boldsymbol{Z}, \boldsymbol{H}_1, \cdots, \boldsymbol{H}_k) && \in \mathbb{R}^{n\times(k+1)\times c}\\
\boldsymbol{S} &= \sigma(\boldsymbol{H}\boldsymbol{s}) && \in \mathbb{R}^{n\times(k+1)\times 1}\\
\widetilde{\boldsymbol{S}} &= \mathrm{reshape}(\boldsymbol{S}) && \in \mathbb{R}^{n\times 1\times(k+1)}\\
\boldsymbol{X}_{out} &= \mathrm{softmax}\left(\mathrm{squeeze}\left(\widetilde{\boldsymbol{S}}\boldsymbol{H}\right)\right) && \in \mathbb{R}^{n\times c},
\end{aligned} \tag{8}$$

where c is the number of node classes. $\boldsymbol{Z} \in \mathbb{R}^{n\times c}$ is the feature matrix derived by applying an MLP network to the original feature matrix. We utilize the symmetrical normalization propagation mechanism $\widehat{\boldsymbol{A}} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$, where $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$. k is a hyperparameter that indicates the depth of the model. $\boldsymbol{s} \in \mathbb{R}^{c\times 1}$ is a trainable projection vector. $\sigma(\cdot)$ is an activation function and we apply *sigmoid*. *stack*, *reshape* and *squeeze* are utilized to rearrange the data dimension so that dimension can be matched during computation.

An illustration of our proposed DAGNN is provided in Figure 5. There are three main steps in DAGNN: transformation, propagation and adaptive adjustment. We first utilize a shared MLP network for feature transformation. Theoretically, MLP can approximate any measurable function [9]. Obviously, $\boldsymbol{Z}$ only contains the information of individual nodes themselves with no structure information included. After transformation, a propagation mechanism $\widehat{\boldsymbol{A}}$ is applied to gather neighborhood information. $\boldsymbol{H}_\ell$ denotes the representations obtained by propagating information from nodes that are ℓ-hop away, hence $\boldsymbol{H}_\ell$ captures the information from the subtree of height ℓ rooted at individual nodes. As the depth ℓ increase, more global information is included in $\boldsymbol{H}_\ell$ because the corresponding subtree is deeper. However, it is difficult to determine a suitable ℓ. Small ℓ may fail to capture sufficient and essential neighborhood information, while large ℓ may bring too much global information and dilute the special local information. Furthermore, each node has a different subtree structure rooted at this node and the most suitable receptive field for each node should be different. To this end, we include an adaptive adjustment mechanism after the propagation. We utilize a trainable projection vector $\boldsymbol{s}$ that is shared by all nodes to generate retainment scores. These scores are used for representations that carry information from various range of neighborhood. These retainment scores measure how much information of the corresponding representations derived by different propagation layers should be retained to generate the final representation for each node. Utilizing this adaptive adjustment mechanism, DAGNN can adaptively balance the information from local and global neighborhoods for each node. Obviously, the transformation process and the adaptive adjustment process have trainable parameters and there is no trainable parameter in the propagation process, leading to a parameter-efficient model. Note that DAGNN is trained end-to-end, which means that these three steps are considered together when optimizing the network.

Decoupling representation transformation from propagation and utilizing the learnable retainment score to adaptively adjust the information from local and global neighborhoods make DAGNN have the ability to generate suitable representations for specific nodes from large and adaptive receptive fields. Besides, removing the entanglement of representation transformation and propagation, we can derive a large neighborhood without introducing more trainable parameters. Also, transforming representations to a low dimensional space at an early stage makes DAGNN computationally

Table 1: Statistics of datasets. The edge density is computed by $\frac{2m}{n^2}$. Note that for a fair comparison with other baselines, we only consider the largest connected component in co-purchase graphs as [27].

Dataset	#Classes	#Nodes	#Edges	Edge Density	#Features	#Training Nodes	#Validation Nodes	#Test Nodes
Cora	7	2708	5278	0.0014	1433	20 per class	500	1000
CiteSeer	6	3327	4552	0.0008	3703	20 per class	500	1000
PubMed	3	19717	44324	0.0002	500	20 per class	500	1000
Coauthor CS	15	18333	81894	0.0005	6805	20 per class	30 per class	Rest nodes
Coauthor Physics	5	34493	247962	0.0004	8415	20 per class	30 per class	Rest nodes
Amazon Computers	10	13381	245778	0.0027	767	20 per class	30 per class	Rest nodes
Amazon Photo	8	7487	119043	0.0042	745	20 per class	30 per class	Rest nodes

Table 2: Results on citation datasets with both fixed and random splits in terms of classification accuracy (in percent).

Models	Cora		CiteSeer		PubMed	
	Fixed	Random	Fixed	Random	Fixed	Random
MLP	61.6 ± 0.6	59.8 ± 2.4	61.0 ± 1.0	58.8 ± 2.2	74.2 ± 0.7	70.1 ± 2.4
ChebNet	80.5 ± 1.1	76.8 ± 2.5	69.6 ± 1.4	67.5 ± 2.0	78.1 ± 0.6	75.3 ± 2.5
GCN	81.3 ± 0.8	79.1 ± 1.8	71.1 ± 0.7	68.2 ± 1.6	78.8 ± 0.6	77.1 ± 2.7
GAT	83.1 ± 0.4	80.8 ± 1.6	70.8 ± 0.5	68.9 ± 1.7	79.1 ± 0.4	77.8 ± 2.1
APPNP	83.3 ± 0.5	81.9 ± 1.4	71.8 ± 0.4	69.8 ± 1.7	80.1 ± 0.2	79.5 ± 2.2
SGC	81.7 ± 0.1	80.4 ± 1.8	71.3 ± 0.2	68.7 ± 2.1	78.9 ± 0.1	76.8 ± 2.6
DAGNN (Ours)	**84.4 ± 0.5**	**83.7 ± 1.4**	**73.3 ± 0.6**	**71.2 ± 1.4**	**80.5 ± 0.5**	**80.1 ± 1.7**

Table 3: Results on co-authorship and co-purchase datasets in terms of classification accuracy (in percent).

Models	Coauthor CS	Coauthor Physics	Amazon Computers	Amazon Photo
LogReg	86.4 ± 0.9	86.7 ± 1.5	64.1 ± 5.7	73.0 ± 6.5
MLP	88.3 ± 0.7	88.9 ± 1.1	44.9 ± 5.8	69.6 ± 3.8
LabelProp	73.6 ± 3.9	86.6 ± 2.0	70.8 ± 8.1	72.6 ± 11.1
LabelProp NL	76.7 ± 1.4	86.8 ± 1.4	75.0 ± 2.9	83.9 ± 2.7
GCN	91.1 ± 0.5	92.8 ± 1.0	82.6 ± 2.4	91.2 ± 1.2
GAT	90.5 ± 0.6	92.5 ± 0.9	78.0 ± 19.0	85.7 ± 20.3
MoNet	90.8 ± 0.6	92.5 ± 0.9	83.5 ± 2.2	91.2 ± 1.3
GraphSAGE-mean	91.3 ± 2.8	93.0 ± 0.8	82.4 ± 1.8	91.4 ± 1.3
GraphSAGE-maxpool	85.0 ± 1.1	90.3 ± 1.2	N/A	90.4 ± 1.3
GraphSAGE-meanpool	89.6 ± 0.9	92.6 ± 1.0	79.9 ± 2.3	90.7 ± 1.6
DAGNN (Ours)	**92.8 ± 0.9**	**94.0 ± 0.6**	**84.5 ± 1.2**	**92.0 ± 0.8**

efficient and memory-saving. In order to compute $\widetilde{A}^{\ell}Z$ efficiently, we choose to compute it sequentially from right to left with time complexity $O(n^2c)$, which saves the computational cost compared to $O(n^3)$ of calculating $\widetilde{A}^{\ell}$ first.

Notably, there are no fully-connected layers utilized as a classifier at the end of this model. In our DAGNN, the final representations $\boldsymbol{X}_{out}$ are used as the final prediction. Thus, the cross-entropy loss for all labeled examples can be calculated as

$$\mathcal{L} = -\sum_{i \in V_L} \sum_{p=1}^{c} Y_{[i,p]} \ln \boldsymbol{X}_{out[i,p]}, \tag{9}$$

where V_L is the set of labeled nodes and $Y \in \mathbb{R}^{n \times c}$ is the label indicator matrix. c is the number of classes.

5 EXPERIMENTAL STUDIES

In this section, we conduct extensive experiments on node classification tasks to evaluate the superiority of our proposed DAGNN. We begin by introducing datasets and experimental setup we utilized. We then compare DAGNN with prior state-of-the-art baselines to demonstrate the effectiveness of DAGNN. Also, we deploy some performance studies to further verify the proposed model.

5.1 Datasets and Setup

We conduct experiments on 7 datasets based on citation, co-authorship, or co-purchase graphs for semi-supervised node classification tasks; those are Cora [25], CiteSeer [25], PubMed [25], Coauthor CS [27], Coauthor Physics [27], Amazon Computers [27], and Amazon Photo [27]. The statistics of these datasets are summarized in Table 1. The detailed description of these datasets are provided in Appendix A.7.

We implemented our proposed DAGNN and some necessary baselines using Pytorch [23] and Pytorch Geometric [4], a library for deep learning on irregularly structured data built upon Pytorch. We consider the following baselines: Logistic Regression (LogReg), Multilayer Perceptron (MLP), Label Propagation (LabelProp) [1], Normalized Laplacian Label Propagation (LabelProp NL) [1], ChebNet [3], Graph Convolutional Network (GCN) [10], Graph Attention Network(GAT) [29], Mixture Model Network (MoNet) [20], GraphSAGE [8], APPNP [11], and SGC [30]. We aim to provide a rigorous and fair comparison between different models on each dataset by using the same dataset splits and training procedure. We tune hyperparameters for all models individually and some baselines even achieve better results than their original reports. For our DAGNN, we tune the following hyperparameters: (1) k ∈ {5, 10, 20}, (2) weight decay ∈ {0, 2e-2, 5e-3, 5e-4, 5e-5}, and (3) dropout rate ∈ {0.5, 0.8}. Our code is publicly available [1].

5.2 Overall Results

The results on citation datasets are summarized in Table 2. To ensure a fair comparison, we use 20 labeled nodes per class as the training set, 500 nodes as the validation set, and 1000 nodes as the test set for all models. For each model, we conduct 100 runs for the fixed training/validation/test split from [10], which is commonly used to evaluate performance by the community. Also, we conduct 100 runs for each model on randomly training/validation/test splits, where we additionally ensure uniform class distribution on the

[1] https://github.com/divelab/DeeperGNN

Table 4: Results with different training set sizes on Cora in terms of classification accuracy (in percent). Results in brackets are the improvements of DAGNN over GCN.

#Training nodes per class	1	2	3	4	5	10	20	30	40	50	100
MLP	30.3	35.0	38.3	40.8	44.7	53.0	59.8	63.0	64.8	65.4	64.0
GCN	34.7	48.9	56.8	62.5	65.3	74.3	79.1	80.8	82.2	82.9	84.7
GAT	45.3	58.8	66.6	68.4	70.7	77.0	80.8	82.6	83.4	84.0	86.1
APPNP	44.7	58.7	66.3	71.2	74.1	79.0	81.9	83.2	83.8	84.3	85.4
SGC	43.7	59.2	67.2	70.4	71.5	77.5	80.4	81.3	81.9	82.1	83.6
DAGNN (Ours)	**58.4**(23.7↑)	**67.7**(18.8↑)	**72.4**(15.6↑)	**75.5**(13.0↑)	**76.7**(11.4↑)	**80.8**(6.5↑)	**83.7**(4.6↑)	**84.5**(3.7↑)	**85.6**(3.4↑)	**86.0**(3.1↑)	**87.1**(2.4↑)

train split as [4]. We compute the average test accuracy of 100 runs. As shown in Table 2, our DAGNN model performs better than the representative baselines by significant margins. Also, the fact that DAGNN achieves state-of-the-art performance on random splits demonstrates the strong robustness of DAGNN. Quantitatively, for the randomly split data, the improvements of DAGNN over GCN are 4.6%, 3.0%, and 3.0% on Cora, CiteSeer, and PubMed, respectively.

The results on co-authorship and co-purchase datasets are summarized in Table 3. We utilize 20 labeled nodes per class as the training set, 30 nodes per class as the validation set, and the rest as the test set. The results of baselines are obtained from [27]. For DAGNN, we conduct 100 runs for randomly training/validation/test splits as [27] to ensure a fair comparison with baselines. Our DAGNN model achieves better performance over the current state-of-the-art models by significant margins of 1.5%, 1.0%, 1.0%, and 0.6% on the Coauthor CS, Coauthor Physics, Amazon Computers, and Amazon Photo, respectively. Note that DAGNN reduces the error rate by 11% on average.

In summary, our DAGNN achieves superior performance on all these seven datasets, which significantly demonstrates the effectiveness of our proposed model. These results verify the superiority of learning node representations from large and adaptive receptive fields, which is achieved by decoupling transformation from propagation and utilizing an adaptive adjustment mechanism in DAGNN.

5.3 Training Set Sizes

The number of training samples is usually limited in the real world graph. Hence, it is necessary to explore how models perform with different training set sizes. To further demonstrate the advantage that our DAGNN is capable of capturing information from large and adaptive receptive fields, we conduct experiments with different training set sizes for several representative baselines. MLP only utilizes node features to learn representations. GCN and GAT include the structure information by propagation representations through edges, however, only limited receptive fields can be taken into consideration and it is shown in Section 3.2 that performance degrades when stacking multiple GCN layers to enable large receptive fields. APPNP, SGC, and our DAGNN all have the ability to deploy a large receptive field. Note that for a fair comparison, we set the depth in APPNP, SGC, and DAGNN as 10, where information in the 10-hop neighborhood can be included. For each model, we conduct 100 runs on randomly training/validation/test splits for every training set size on Cora dataset. The results are present in Table 4 where our improvements over GCN are also highlighted. Our DAGNN

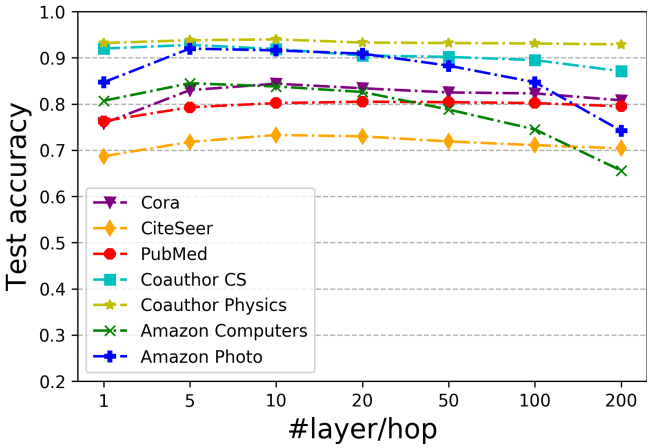

Figure 6: Results of DAGNN with different depths.

achieves the best performance under different training set sizes. Notably, The superiority of our DAGNN can be demonstrated more obviously when fewer training nodes are used. The improvement of DAGNN over GCN increases greatly when the number of training nodes decreases. Extremely, only utilizing one training node per class, DAGNN achieves an overwhelming result over GCN by a significant margin of 23.7%. These considerable improvements are mainly attributed to the advantage that DAGNN can incorporate information from large receptive fields by removing the entanglement of representation transformation and propagation. Reachable large receptive fields are beneficial for propagating training signals to distant nodes, which is very essential when the number of training nodes is limited. APPNP and SGC also can gather information from a large neighborhood, but they perform not as good as DAGNN. This performance gap is mainly caused by the adaptive adjustment of DAGNN, which can adjust the information from different receptive fields for each node adaptively.

5.4 Model Depths

In order to investigate when over-smoothing happens in our DAGNN, we conduct experiments for DAGNN with different depths. For each dataset, we choose different hyperparameter k in DAGNN, which means the k-hop neighborhood is visible by each node, and conduct 100 runs for each setting. The results are illustrated in Figure 6. For citation and co-authorship datasets, very deep models with large numbers of propagation iterations can be applied with keeping stable or slightly decreasing performance, which can be attributed to

the design that we decouple the transformation from propagation and utilize adaptive receptive fields to learn node representations in DAGNN. Note that performances on co-purchase datasets decrease obviously with the increment of depth. This should be resulted by their larger value of edge density than other datasets, as shown in Table 1. Intuitively, when nodes are more densely connected, their representations will become indistinguishable by applying a smaller number of propagation iterations, which aligns with our assumption in Section 3.3 that further connection exists between the graph topology information and convergence speed.

6 CONCLUSION

In this paper, we consider the performance deterioration problem existed in current deep graph neural networks and develop new insights towards deeper graph neural networks. We first provide a systematical analysis on this issue and argue that the key factor that compromises the network performance is the entanglement of representation transformation and propagation. We propose to decouple these two operations and show that deep graph neural networks without this entanglement can leverage large receptive fields without suffering from performance deterioration. Further, we provide a theoretically analysis of the above strategy when building very deep models, which can serve as a rigorous and gentle description of the over-smoothing issue. Utilizing our insights, DAGNN is proposed to conduct node representation learning with the ability to capture information from large and adaptive receptive fields. According to our comprehensive experiments, our DAGNN achieves a better performance than current state-of-the-art models by significant margins, especially when training samples are limited, which demonstrates its superiority.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grants DBI-1922969, IIS-1908166, and IIS-1908198.

REFERENCES

[1] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. 2009. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks* 20, 3 (2009), 542–542.

[2] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. In *Thirty-Four AAAI Conference on Artificial Intelligence*.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.

[4] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[5] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. In *International Conference on Machine Learning*. 2083–2092.

[6] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1416–1424.

[7] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*. 1263–1272.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[9] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. 1989. Multilayer feedforward networks are universal approximators. *IEEE Transactions on Neural Networks* 2, 5 (1989), 359–366.

[10] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

[11] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*.

[12] Panqanamala Ramana Kumar and Pravin Varaiya. 2015. *Stochastic systems: Estimation, identification, and adaptive control*. Vol. 75. SIAM.

[13] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-Attention Graph Pooling. In *International Conference on Machine Learning*. 3734–3743.

[14] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[15] Meng Liu, Zhengyang Wang, and Shuiwang Ji. 2020. Non-Local Graph Neural Networks. *arXiv preprint arXiv:2005.14612* (2020).

[16] László Lovász et al. 1993. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty* 2, 1 (1993), 1–46.

[17] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 723–731.

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

[19] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 43–52.

[20] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5115–5124.

[21] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*. 807–814.

[22] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web*. Technical Report. Stanford InfoLab.

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *Proceedings of Neural Information Processing Systems Autodiff Workshop*.

[24] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*.

[25] 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.

[26] Eugene Seneta. 2006. *Non-negative matrices and Markov chains*. Springer Science & Business Media.

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

[28] Gabriel Taubin. 1995. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM, 351–358.

[29] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representation*.

[30] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning*. 6861–6871.

[31] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *International Conference on Learning Representations*.

[32] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *International Conference on Machine Learning*. 5449–5458.

[33] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*. 4800–4810.

[34] Hao Yuan and Shuiwang Ji. 2020. StructPool: Structured Graph Pooling via Conditional Random Fields. In *International Conference on Learning Representations*.

[35] Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*. 5165–5175.

[36] Muhan Zhang and Yixin Chen. 2017. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 575–583.

[37] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

A APPENDIX

In this section, we provide necessary information for reproducing our insights and experimental results. These include the quantitative results and qualitative visualization on more datasets that can further support our insights, the proofs of lemmas, and the detailed description of datasets.

A.1 Test Accuracy and Smoothness Metric Value of GCNs

Test accuracy and smoothness metric value of node representations with different numbers of GCN layers are shown in Figure 7 for CiteSeer and PubMed. They have the same trends as we discussed in Section 3.2.

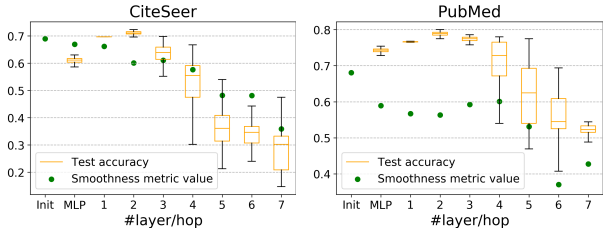

Figure 7: Test accuracy and smoothness metric value of node representations with different numbers of GCN layers on CiteSeer and PubMed. "Init" means the smoothness metric value of the original data.

A.2 Test Accuracy and Smoothness Metric Value of Models as Eq.(6)

Test accuracy and smoothness metric value of node representations with different numbers of layers adopted in models as Eq.(6) are shown in Figure 8 for CiteSeer and PubMed. It is illustrated that after decoupling transformation from propagation, we can apply deeper models without suffering from performance degradation.

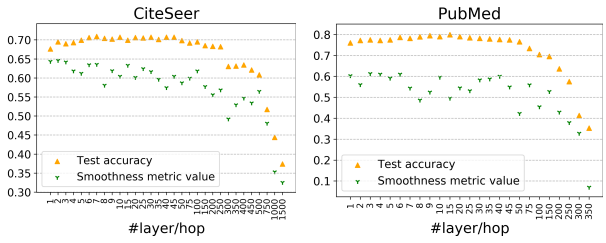

Figure 8: Test accuracy and smoothness metric value of node representations with different numbers of layers adopted in models as Eq.(6) on CiterSeer and PubMed.

A.3 Visualization of Representations Derived by GCNs

The t-SNE visualization of node representations derived by different numbers of GCN layers are shown in Figure 9 and 10 for CiteSeer and PubMed, respectively. The node representations become indistinguishable when several layers are deployed.

A.4 Visualization of Representations Derived by Models as Eq.(6)

The t-SNE visualization of node representations derived by model as Eq.(6) with different numbers of layers are shown in Figure 11 and 12 for CiteSeer and PubMed, respectively. It is shown that after the entanglement of representation transformation and propagation is removed, the model with a large receptive field, such as 50-hop, still generating distinguishable node representations. The over-smoothing issue affects the distinguishability only when an extremely receptive field, like 200-hop, is adopted.

A.5 Proof for Lemma 3.3

Proof. If λ is an eigenvalue of $\widehat{\boldsymbol{A}}_{\oplus}$ with left eigenvector $\boldsymbol{v}_l$ and right eigenvector $\boldsymbol{v}_r$, we have $\boldsymbol{v}_l\widehat{\boldsymbol{A}}_{\oplus} = \lambda\boldsymbol{v}_l$ and $\widehat{\boldsymbol{A}}_{\oplus}\boldsymbol{v}_r = \lambda\boldsymbol{v}_r$, i.e. $\boldsymbol{v}_l\widetilde{\boldsymbol{D}}^{-1}\widetilde{\boldsymbol{A}} = \lambda\boldsymbol{v}_l$ and $\widetilde{\boldsymbol{D}}^{-1}\widetilde{\boldsymbol{A}}\boldsymbol{v}_r = \lambda\boldsymbol{v}_r$. We right multiply the first eigenvalue equation with $\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$ and left multiply the second eigenvalue equation with $\widetilde{\boldsymbol{D}}^{\frac{1}{2}}$, respectively. Then we can derive $(\boldsymbol{v}_l\widetilde{\boldsymbol{D}}^{-\frac{1}{2}})\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}} = \lambda(\boldsymbol{v}_l\widetilde{\boldsymbol{D}}^{-\frac{1}{2}})$ and $\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}(\widetilde{\boldsymbol{D}}^{\frac{1}{2}}\boldsymbol{v}_r) = \lambda(\widetilde{\boldsymbol{D}}^{\frac{1}{2}}\boldsymbol{v}_r)$. Hence, λ is also an eigenvalue of $\widehat{\boldsymbol{A}}_{\odot}$ with left eigenvector $\boldsymbol{v}_l\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$ and right eigenvector $\widetilde{\boldsymbol{D}}^{\frac{1}{2}}\boldsymbol{v}_r$. From $\widehat{\boldsymbol{A}}_{\odot}$ to $\widehat{\boldsymbol{A}}_{\oplus}$, we can prove it in the same way. □

A.6 Proof for Lemma 3.4

Proof. We first prove that $\widehat{\boldsymbol{A}}_{\oplus}$ and $\widehat{\boldsymbol{A}}_{\odot}$ always have an eigenvalue 1 and all eigenvalues λ satisfy $|\lambda| \leq 1$. We have $\widehat{\boldsymbol{A}}_{\oplus}\boldsymbol{e}^T = \boldsymbol{e}^T$ because each row of $\widehat{\boldsymbol{A}}_{\oplus}$ sums to 1. Therefore, 1 is an eigenvalue of $\widehat{\boldsymbol{A}}_{\oplus}$. Suppose that there exists an eigenvalue λ that $|\lambda| > 1$ with eigenvector $\boldsymbol{v}$, then the length of the right side in $\widehat{\boldsymbol{A}}^k_{\oplus}\boldsymbol{v} = \lambda^k\boldsymbol{v}$ grows exponentially when k goes to infinity. This indicates that some entries of $\widehat{\boldsymbol{A}}^k_{\oplus}$ shoulde be larger than 1. Nevertheless, all entries of $\widehat{\boldsymbol{A}}^k_{\oplus}$ are positive and each row of $\widehat{\boldsymbol{A}}^k_{\oplus}$ always sums to 1, hence no entry of $\widehat{\boldsymbol{A}}^k_{\oplus}$ can be larger than 1, which leads to contradiction. From Lemma 3.3, $\widehat{\boldsymbol{A}}_{\oplus}$ and $\widehat{\boldsymbol{A}}_{\odot}$ have the same eigenvalues. Therefore, $\widehat{\boldsymbol{A}}_{\oplus}$ and $\widehat{\boldsymbol{A}}_{\odot}$ always have an eigenvalue 1 and all eigenvalues λ satisfy $|\lambda| \leq 1$.

According to the Perron-Frobenius Theorem for Primitive Matrices [26], there exists an eigenvalue r for an $n \times n$ non-negative primitive matrix such that $r > |\lambda|$ for any eigenvalue $\lambda \neq r$ and the eigenvectors associated with r are unique. The property that the given graph is connected can guarantee that for $\forall i, j$: $\exists k$ s.t. $\widehat{\boldsymbol{A}}^k_{\oplus}[i, j] > 0$. Furthermore, there must exsit some k that can make all entries of $\widehat{\boldsymbol{A}}^k$ to be simultaneously positive because self-loops are included in the graph. Formally, $\exists k$: $\widehat{\boldsymbol{A}}^k_{\oplus}[i, j] > 0$ for $\forall i, j$. Hence, $\widehat{\boldsymbol{A}}_{\oplus}$ is a non-negative primitive matrix. From the Perron-Frobenius Theorem for Primitive Matrices, $\widehat{\boldsymbol{A}}_{\oplus}$ always has an eigenvalue 1 with unique associated eigenvectors and all other eigenvalues λ satisfy $|\lambda| < 1$. Based on Lemma 3.3, this property can be extended

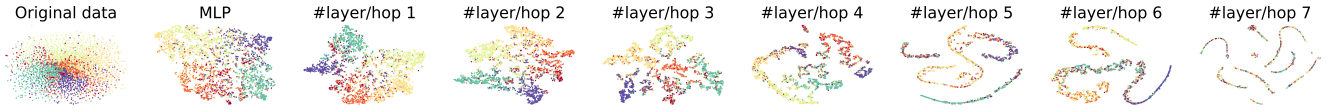

Figure 9: t-SNE visualization of node representations derived by different numbers of GCN layers on CiteSeer. Colors represent node classes.

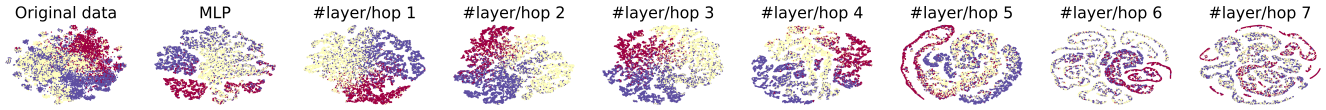

Figure 10: t-SNE visualization of node representations derived by different numbers of GCN layers on PubMed. Colors represent node classes.

Figure 11: t-SNE visualization of node representations derived by models as Eq.(6) with different numbers of layers on CiteSeer. Colors represent node classes.

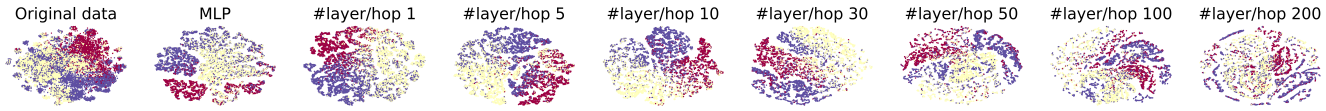

Figure 12: t-SNE visualization of node representations derived by models as Eq.(6) with different numbers of layers on PubMed. Colors represent node classes.

to $\widehat{\boldsymbol{A}}_{\odot}$. We then compute the eigenvectors associated with eigenvalue 1. Obviously, $\boldsymbol{e}^T$ is the right eigenvector of $\widehat{\boldsymbol{A}}_{\oplus}$ associated with eigenvalue 1. Next, assume $\boldsymbol{v}_l$ is the left eigenvector of $\widehat{\boldsymbol{A}}_{\oplus}$ associated with eigenvalue 1 and thus $\boldsymbol{v}_l\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$ is the left eigenvector of $\widehat{\boldsymbol{A}}_{\odot}$ associated with eigenvalue 1. We know $\widehat{\boldsymbol{A}}_{\odot}$ is a symmetric matrix, whose left and right eigenvectors associated with the same eigenvalue are simply each other's transpose. Hence, we utilize $\boldsymbol{v}_l\widetilde{\boldsymbol{D}}^{-\frac{1}{2}} = (\widetilde{\boldsymbol{D}}^{\frac{1}{2}}\boldsymbol{e}^T)^T$ to obtain $\boldsymbol{v}_l = \boldsymbol{e}\widetilde{\boldsymbol{D}}$. After deriving the eigenvectors of $\widehat{\boldsymbol{A}}_{\oplus}$ associated with eigenvalue 1, corresponding eigenvectors of $\widehat{\boldsymbol{A}}_{\odot}$ can be computed by Lemma 3.3. □

A.7 Datasets Description and Statistics

Citation datasets. Cora, CiteSeer and PubMed [25] are representative citation network datasets where nodes and edges denote documents and their citation relationships, respectively. Node features are formed by bay-of-words representations for documents. Each node has a label indicating what field the corresponding document belongs to.

Co-authorship datasets. Coauthor CS and Coauthor Physics [27] are co-authorship graphs datasets. Nodes denote authors, which are connected by an edge if they co-authored a paper. Node features represent paper keywords for each author's papers. Each node has a label denoting the most active fields of study for the corresponding author.

Co-purchase datasets. Amazon Computers and Amazon Photo [27] are segments of the Amazon co-purchase graph [19] where nodes are goods and edges denote that two goods are frequently bought together. Node features are derived from bag-of-words representations for product reviews and class labels are given by the product category.

The datasets statistics are summarized in Table 1.