# DrGNN: Deep Residual Graph Neural Network with Contrastive Learning

**Anonymous authors**
**Paper under double-blind review**

## Abstract

Recent studies reveal that deep representation learning models without proper regularization can suffer from the *dimensional collapse* problem, i.e., representation vectors span over a lower dimensional space. In the domain of graph deep representation learning, the phenomenon that the node representations are indistinguishable and even shrink to a constant vector is called *oversmoothing*. Based on the analysis of the rank of node representations, we find that representation oversmoothing and dimensional collapse are highly related to each other in deep graph neural networks, and the oversmoothing problem can be interpreted by the dimensional collapse of the node representation matrix. Then, to address the dimensional collapse and the oversmoothing together in deep graph neural networks, we first find vanilla *residual connections* and *contrastive learning* producing sub-optimal outcomes by ignoring the structured constraints of graph data. Motivated by this, we propose a novel graph neural network named DrGNN to alleviate the oversmoothing issue from the perspective of addressing dimensional collapse. Specifically, in DrGNN, we design a topology-preserving residual connection for graph neural networks to force the low-rank of hidden representations close to the full-rank input features. Also, we propose the structure-guided contrastive learning to ensure only close neighbors who share similar local connections can have similar representations. Empirical experiments on multiple real-world datasets demonstrate that DrGNN outperforms state-of-the-art deep graph representation baseline algorithms.

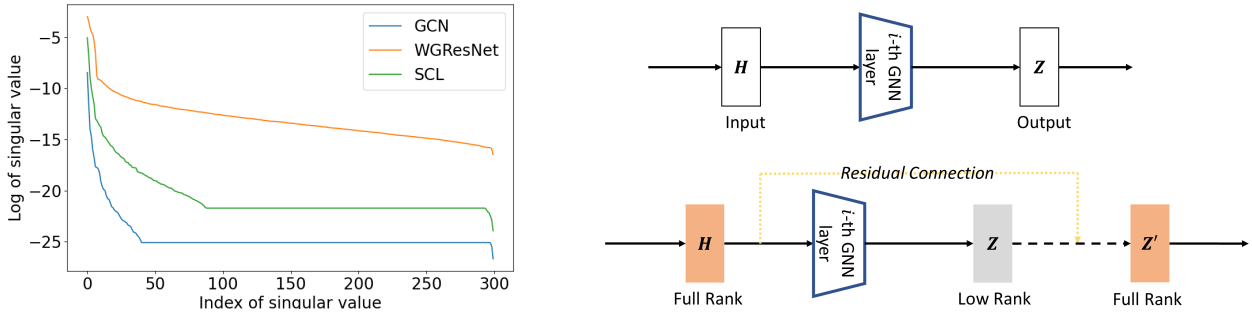
## 1 Introduction

Representation learning models have achieved outstanding performance for various application domains by outputting informative hidden representations, such as computer vision and natural language processing. Recent studies (Hua et al., 2021; Jing et al., 2022; Guo et al., 2023) show that the deep representation learning models without proper regularization tend to produce representations that collapse along certain directions, known as the *dimensional collapse*, which can be further interpreted by the visualization of the singularity ranking of the matrices of representations (Hua et al., 2021).

With the advent of big data, graph structures recently received increasing research attention for their ability to encode complex interactions. Similarly, the deep representation learning models on graphs are also found affected by representation issues. Especially, the node representation vectors outputted by *deeper* graph neural networks are not discriminative from each other and directly hurt the performance of node classification and link prediction tasks and their corresponding applications. This phenomenon is called *oversmoothing* in the graph representation learning domain (Li et al., 2018; Oono & Suzuki, 2020; Rusch et al., 2023).

To begin with, we find that the oversmoothing in graph deep learning can be interpreted by dimensional collapse from the low rank of the representation matrix. A detailed theoretical derivation can be found in Appendix A. To empirically demonstrate the existence of dimensional collapse in deep graph neural networks, we conduct a toy experiment on the Cora benchmark dataset (Lu & Getoor, 2003) by exploring the rank of the covariance matrix of the node representations. The analysis is visualized in Figure 1, where the $x$-axis is the index of the sorted singular values of the covariance matrix of the representation matrix, and the $y$-axis is the logarithm of the singular value. In Figure 1, we can see that the number of non-zero singular values is much

smaller than the number of representation dimensions for a deep Graph Convolutional Network (GCN) (Kipf & Welling, 2017). This suggests that the representation matrix is *low-rank*, and the discrimination of node representation vectors only relies on a few dimensions, which naturally increases the difficulty of effectively discriminating node presentations and makes tasks (e.g., node classification and link prediction) groundless.




(a) (b)

Figure 1: (a). A toy example on the Cora dataset demonstrates the rank deficiency, where GCN is chosen as the backbone, the number of layers is set to 64, and the dimension of representation is 100. WGResNet and SCL are two major components of our proposed DrGNN, which are model-agnostic and can be added on GCN for example. (b) The visualization about how vanilla residual connections of neural layers turn the low-rank representation $\boldsymbol{Z}$ into a full-rank representation $\boldsymbol{Z}'$.

However, stacking more graph neural layers is indispensable in some cases (e.g., heterophily, long-range interaction, etc.). For example, for a certain query node, the close neighbors can have missing features and labels; then, we may need the information from distant neighbors to represent the query node, as two case studies show in Section 3.2. According to the message passing of graph neural networks Xu et al. (2018), adding one graph neural network layer equals aggregating information from one-hop neighbors, such that we need to stack multiple graph neural network layers. **Thus, how to stack more graph neural network layers to maintain and even boost the representation power motivates our paper**.

Addressing the dimensional collapse problem for deep neural networks, previous study of residual connections (He et al., 2016) among neural layers can be an effective manner, i.e., it has been discovered that residual connections across neural network layers force the low-rank of hidden representations close to the full-rank input features (Jing et al., 2022), as shown in Figure 1 (b). The residual connections pave the way for eliminating the dimensional collapse for deep neural networks and indicate the de-oversmoothing probability for deep graph neural networks, but for non-IID graph data, the vanilla residual connections ignore the topological assumption of graph data that closer neighbors are more important during the embedding process. Formally, adding vanilla residual connection can induce the drawback of what we call the "*shading neighbors*" effects, i.e., close neighbors become less important during the neural representation process. The details are discussed in Section 2.2.

Moreover, for the oversmoothing phenomenon in the graph representation learning domain, the direct result is that individual representations are indistinguishable. Hence, contrastive learning serves as a viable solution, but the existing work (Guo et al., 2023) simply introduces vanilla contrastive loss as a regularization while failing to consider the topological relationship of positive and negative pairs.

Facing the latent dimensional collapse problem (i.e., non-sufficient singular values of covariance matrix of representations) and observable oversmoothing problem (i.e., non-discriminative node embedding vectors), we propose two effective solutions, i.e., Weight-Decaying Graph Residual Connection (WG-ResNet) and Structure-Guided Contrastive Learning (SCL) for deep graph neural networks. In brief, WG-ResNet applies weighted residual connections to preserve the input graph topology, and SCL also weighs different positive and negative pairs based on their topological relations. Besides the theoretical analysis, a visualization of SCL and WG-ResNet in addressing dimensional collapse is also shown in Figure 1 (a). Moreover, it can be observed that SCL itself can alleviate the dimensional collapse to some extent, i.e., alleviating oversmoothing by contrastive learning can alleviate the dimensional collapse, which suggests the close relationship between dimensional collapse and oversmoothing.

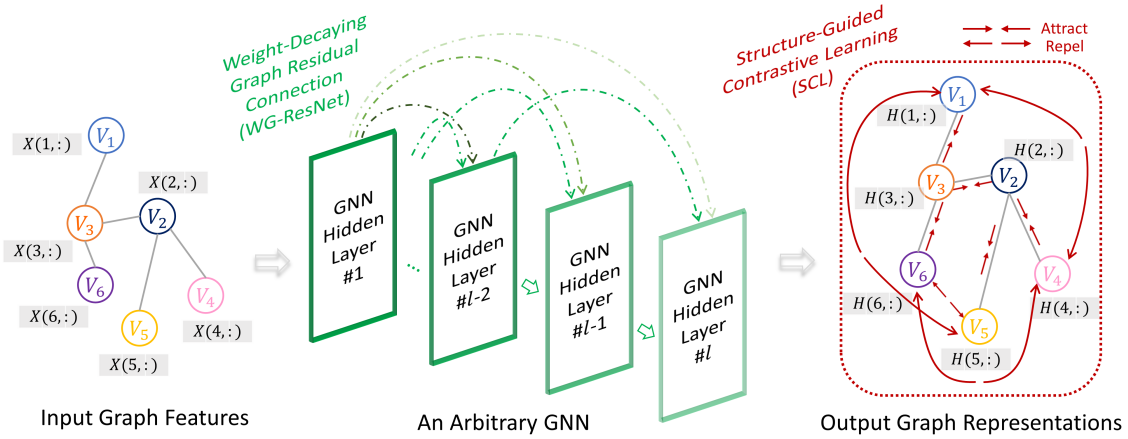



Figure 2: An Arbitrary GNN Backbone with the Proposed DrGNN.

Hence, we propose an end-to-end graph neural network model DrGNN, which encloses SCL and WG-ResNet in a model-agnostic manner to help arbitrary graph neural networks go deeper effectively compared to state-of-the-art baselines, supported by theoretical and empirical analysis. Furthermore, we designed extensive ablation studies to show that SCL and WG-ResNet both contribute to boosting the representation power, and their combination can reach optimal results in terms of node classification.

## 2 The Proposed DrGNN

In this section, we begin with the overview of DrGNN and then provide the details of the Weight-decaying Graph Residual Connection (WG-ResNet) and Structure-guided Contrastive Learning (SCL). We formalize the problem of graph embedding within the context of an undirected graph $G = (V, E, \boldsymbol{X})$, where $V$ consists of $n$ nodes, $E$ consists of $m$ edges, $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ denotes the feature matrix and $d$ is the feature dimension. We let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ denote the adjacency matrix and denote $\boldsymbol{A}_i \in \mathbb{R}^n$ as the adjacency vector for node $v_i$. $\boldsymbol{H}_i \in \mathbb{R}^h$ is the hidden representation vector of $v_i$.

### 2.1 Overview

The overview of our proposed DrGNN is shown in Figure 2 and DrGNN consists of two parts, including the graph architecture WG-ResNet and contrastive loss SCL. Specifically, the green dash line stands for WG-ResNet, where $\boldsymbol{H}^{(l)}$ at the $l$-th layer will be adjusted by its second last layer $\boldsymbol{H}^{(l-2)}$ and the first layer $\boldsymbol{H}^{(1)}$ with proper weights. The red dash line in Figure 2 stands for SCL, where we sample positive node pairs and negative node pairs based on the input graph topology such that the hidden representations of positive node pairs get closer and negative ones are pushed farther apart. The overall of DrGNN in terms of loss functions and architectures is expressed as follows.

$$\mathcal{L}_{\text{DrGNN}} = \mathcal{L}_{GNN} + \alpha \mathcal{L}_{\text{SCL}} \tag{1}$$

where $\mathcal{L}_{GNN}$ denotes the loss of the downstream task (e.g., node classification) using an arbitrary GNN model (e.g., GCN (Kipf & Welling, 2017)) equipped with WD-ResNet, $\mathcal{L}_{\text{SCL}}$ is the structure-guided contrastive loss function, and $\alpha$ is a constant hyperparameter. The details of WG-ResNet (Section 2.2) and SCL (Section 2.3) are introduced below.

### 2.2 Weight-Decaying Graph Residual Connection (WG-ResNet)

As shown in Figure 1 (b), the vanilla residual connections (e.g., ResNet (He et al., 2016)) have the potential to alleviate the dimensional collapse of deep neural networks. But for deep graph neural networks, we discover that simply adding residual connections leads to the sub-optimal solution. As the way of ResNet stacking graph neural network layers, the importance of close neighbors' features gradually decreases during the GNN

information aggregation process, and the faraway neighbor information becomes dominant. More concretely speaking, because the residual connection of ResNet connects the current neural layer with its second last layer, taking graph convolutional network (GCN) (Kipf & Welling, 2017) as an example, the vanilla residual connection of ResNet is expressed as follows.

$$\boldsymbol{H}^{(l)} = \mathrm{RELU}(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l-1)}) + \boldsymbol{H}^{(l-2)} \tag{2}$$

where $l(\geq 2)$ denotes the index of layers, $\boldsymbol{H}^{(l-1)}$ and $\boldsymbol{H}^{(l-2)}$ are the hidden representations at corresponding layers, RELU is the activation function, $\boldsymbol{W}^{(l-1)}$ is the learnable weight matrix, and $\hat{\boldsymbol{A}}$ is the re-normalized self-looped adjacency matrix with $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ and $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$, where $\tilde{\boldsymbol{D}}$ is the degree matrix. Without loss of generality, we can assume the last layer of GNNs $l$ is divisible by 2. Then, just by extending the dangling $\boldsymbol{H}^{(l-t)}$ iteratively (e.g., first substituting $\boldsymbol{H}^{(l-2)}$ with its previous residual blocks), the above Eq. 2 for the standard residual connection (i.e., the way of ResNet He et al. (2016)) could be rewritten as follows.

$$\begin{aligned}\boldsymbol{H}^{(l)} &= \mathrm{RELU}(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l-1)}) + \mathrm{RELU}(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l-3)}\boldsymbol{W}^{(l-3)}) + \boldsymbol{H}^{(l-4)} \\ &= \underbrace{\mathrm{RELU}(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l-1)}) + \mathrm{RELU}(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l-3)}\boldsymbol{W}^{(l-3)}) + \cdots}_{\text{Information aggregated from the faraway neighbors}} + \underbrace{\mathrm{RELU}(\hat{\boldsymbol{A}}\boldsymbol{H}^{(i)}\boldsymbol{W}^{(i)}) + \cdots + \mathrm{RELU}(\hat{\boldsymbol{A}}\boldsymbol{H}^{(1)}\boldsymbol{W}^{(1)})}_{\text{Information aggregated from the nearest neighbors}}\end{aligned} \tag{3}$$

According to (Xu et al., 2019), stacking $l$ layers in GNNs and obtaining $\boldsymbol{H}^{(l)}$ can be interpreted as aggregating information from $l$-hop neighbors for node hidden representations. As shown in Eq. 3, when we stack more layers in GNNs, the information collected from faraway neighbors becomes dominant (as there are more terms regarding the information from faraway neighbors), and dilutes the information collected from the nearest neighbors (e.g., 1-hop or 2-hop neighbors). This phenomenon contradicts the general intuition that the close neighbors of a node carry the most important information, and the importance degrades with faraway neighbors. Formally, we describe this phenomenon as *shading neighbors* effect when stacking graph neural layers, as the importance of the nearest neighbors is diminishing. As the enlarging the field of information aggregation, this way cannot essentially get rid of the overshooting problem. We empirically demonstrate that the *shading neighbors* effect degrades GNN performance in downstream tasks in Section 3.5: Specifically, we show that (1) vanilla residual connection of ResNet exhibits the *shading neighbors* effect in graph representation learning; (2) jumping knowledge (Xu et al., 2018) can be a viable solution to mitigate the *shading neighbors* effect to some extent; (3) our proposed WG-ResNet achieves the best effectiveness in addressing the *shading neighbors* effect.

To formally introduce our proposed generic graph architecture, i.e., **Weight-Decaying Graph Residual Connection** (WG-ResNet), we first introduce the formulation and then provide insights regarding why it can work. Specifically, our WG-ResNet introduces the layer similarity and weight decaying factor as follows.

$$\boldsymbol{H}^{(l)} = e^{\cos(\boldsymbol{H}^{(1)},\ \tilde{\boldsymbol{H}}^{(l)}) - l/\lambda} \cdot \tilde{\boldsymbol{H}}^{(l)} + \boldsymbol{H}^{(l-2)}, \text{ and } \tilde{\boldsymbol{H}}^{(l)} = \mathrm{RELU}(\hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l-1)}) \tag{4}$$

where $\cos(\boldsymbol{H}^{(1)}, \tilde{\boldsymbol{H}}^{(l)}) = \frac{1}{n}\sum_i \frac{\boldsymbol{H}_i^{(1)}(\tilde{\boldsymbol{H}}_i^{(l)})^\top}{\|\boldsymbol{H}_i^{(1)}\|\|\tilde{\boldsymbol{H}}_i^{(l)}\|}$ measures the similarity between the $l$-th layer and the 1-st layer, and we use the exponential function $e(\cdot)$ to map the cosine similarity ranging from $[-1, 1]$ to $[e^{-1},\ e^{1}]$ to avoid the negative similarity weights. Moreover, the term $e^{-l/\lambda}$ is the decaying factor to further adjust the similarity weight of $\tilde{\boldsymbol{H}}^{(l)}$, where $\lambda$ is a constant hyperparameter.

Our introduced similarity $e^{\cos(\boldsymbol{H}^{(1)},\ \tilde{\boldsymbol{H}}^{(l)}) - l/\lambda}$ is not fixed but dynamic during each training iteration to reflect the optimized similarity of different layers, which also expands the hypothesis space of deeper GNNs. More importantly, the introduced decaying factor $e^{-l/\lambda}$ aims to mitigate the *shading neighbor effect* by strengthening layer-wise dependency and preserving topology information in deeper GNNs. As $\lambda$ is a positive constant, the value of $e^{-l/\lambda}$ is decreasing as $l$ increases, such that the later stacked layers become less influential than the previously stacked ones due to the decaying weight. In contrast, without the decaying factor, the layer-wise weights remain independent, and the shading neighbors effect persists. Moreover, we visualize the layer-wise weight distribution of different residual connection methods and their effectiveness in Appendix B. From another perspective, the hyperparameter $\lambda$ of the decaying factor actually controls

the number of effective neural layers in deeper GNNs, i.e., *after the effective layer, stacking more layers just maintains the performance or brings marginal upgrade.* We find its value directly related to the diameter of the input graph, and this finding is quite important to avoid heavy hyperparameter tuning. The detailed discussion can be found in Section 3.4.

### 2.3 Structure-Guided Contrastive Learning (SCL)

According to (Hua et al., 2021; Jing et al., 2022), contrastive representation learning methods show success in preventing dimensional collapse for image recognition. For graph data, the contrastive methods are able to construct the positive and negative sets and minimize the similarity of the negative pairs (Wang & Isola, 2020; Guo et al., 2023). However, simply adopting the idea of contrastive regularization in deep graph neural networks could not fully alleviate the oversmoothing issue due to ignoring the topological relation of non-IID graph data. To address this issue with the geometry consideration, we propose the **Structure-guided Contrastive Learning** (SCL) as follows. Intuitively, for an anchor node $v_i$, its positive sample (i.e., connected node) should have close node representations, and its negative sample (i.e., disconnected node) should have discriminative node representations. Moreover, we propose $\sigma$ and $\gamma$ to distinguish the importance of positive and negative samples based on the input structure.

$$\mathcal{L}_{\text{SCL}} = -\mathbb{E}_{v_i \in V}[\mathbb{E}_{v_j \in \mathcal{N}_i}(\sigma_{ij} \log(f(\boldsymbol{z}_i, \boldsymbol{z}_j))) + \mathbb{E}_{v_k \in \bar{\mathcal{N}}_i}(\gamma_{ik} \log(1 - f(\boldsymbol{z}_i, \boldsymbol{z}_k)))]$$
$$\sigma_{ij} = \frac{n^2}{m} \cdot \frac{1 - \text{dist}(\boldsymbol{A}_i, \boldsymbol{A}_j)/n}{\sum_{v_{i'} \in V, v_{j'} \in \mathcal{N}_{i'}} (1 - \text{dist}(\boldsymbol{A}_{i'}, \boldsymbol{A}_{j'})/n)} \text{ and } \gamma_{ik} = \frac{n^2}{n^2 - m} \cdot \frac{1 + \text{dist}(\boldsymbol{A}_i, \boldsymbol{A}_k)/n}{\sum_{v_{i'} \in V, v_{k'} \in \bar{\mathcal{N}}_{i'}} (1 + \text{dist}(\boldsymbol{A}_{i'}, \boldsymbol{A}_{k'})/n)} \tag{5}$$

where the connected edge ($v_i$, $v_j$) forms the positive pair, and the disconnected edge ($v_i$, $v_k$) forms the negative pair. $\boldsymbol{z}_i = g(\boldsymbol{H}_i^{(l)})$, $g(\cdot)$ is an encoder mapping $\boldsymbol{H}_i^{(l)}$, $f(\cdot)$ is a similarity function (e.g., $f(\boldsymbol{a}, \boldsymbol{b}) = e^{\frac{\boldsymbol{a}\boldsymbol{b}^\top}{\|\boldsymbol{a}\| \|\boldsymbol{b}\|}}$), dist$(\cdot)$ is a distance measurement function (e.g., hamming distance (Norouzi et al., 2012)), $\mathcal{N}_i$ is the set containing one-hop neighbors of node $v_i$, $\bar{\mathcal{N}}_i$ is the complement of the set $\mathcal{N}_i$, $m$ is the number of edges and $n$ is the number of vertices. Moreover, ($v_{i'}$, $v_{j'}$) iterates over all connected edges in the input graph, and ($v_{i'}$, $v_{k'}$) iterates over all disconnected edges in the input graph.

The intuition of Eq. 5 is to maximize the similarity of the representations of the positive pairs and to minimize the similarity of the representations of the negative pairs, such that the node representations become discriminative. In which process, some previous research works (Perozzi et al., 2014; Grover & Leskovec, 2016; Le, 2021) would first assume that the importance of each edge is identical. However, such an assumption does not always get satisfied in many applications (Velickovic et al., 2017; Faisal et al., 2015). To address this issue, we reweigh the importance of edges by considering the graph topological structure via importance scores $\sigma$ and $\gamma$. Therefore, for a positive pair, if two nodes have similar topological structures (e.g., ego-networks), the importance score $\sigma$ of this node pair should be large; but for a negative pair, if two nodes also have similar topological structures, the importance score $\gamma$ of this negative pair should be small.

Next, we show how our proposed SCL mitigates the oversmoothing issue with the theoretical analysis. Note that, in the following derivation, we refer the edge $e$ to any possible connection, which can be either an existing edge or a non-existing edge. To be specific, if node $i$ and node $k$ do not connect in the input graph, then $e_{ik}$ represents the non-existing edge between node $i$ and node $k$, denoted as a negative edge.

To begin with, we denote the probability of sampling a connection $e_{ij}$ and it is a positive connection as $\tilde{P}_{pos}(e_{ij}) \propto P(e_{ij}, y = 1)$, i.e., the sampled pair of two nodes $v_i$ and $v_j$ connect in the input undirected graph. Then, $\tilde{P}_{pos}(e_{ij})$ can be further expressed as $\tilde{P}_{pos}(e_{ij}) = \sigma_{ij} P_{pos}(e_{ij})$, where $P_{pos}(e_{ij}) = \frac{2m}{n^2}$ is the prior probability that the sampling is positive, and $\sigma_{ij}$ is the conditional probability for the joint probability $\tilde{P}_{pos}(e_{ij})$. Note that a positive connection (e.g., $e_{ij}$) stands for two connected nodes $v_i$ and $v_j$ forming a positive pair. Similarly, for disconnected two nodes $v_i$ and $v_k$ (i.e., the negative pair or negative connection $e_{ik}$), we denote $\tilde{P}_{neg}(e_{ik}) = \gamma_{ik} P_{neg}(e_{ik})$, where $P_{neg}(e_{ik}) = 1 - \frac{2m}{n^2}$ is the prior probability of sampling a negative connection, and we interpret $\gamma_{ik}$ as the conditional probability for the joint probability $\tilde{P}_{neg}(e_{ik})$. Finally, we denote $\theta$ to be the parameters of the multi-layer GNN model $\mathcal{G}_\theta(\cdot)$, i.e., $\boldsymbol{Z} = \mathcal{G}_\theta(\boldsymbol{A}, \boldsymbol{X})$, such that we can prove that SCL could alleviate the oversmoothing issue from the perspective of generative adversarial network (GAN) (Goodfellow et al., 2014) as follows.

**Proposition 2.1.** *$\mathcal{L}_{SCL}$, the loss function based on contrastive learning, can be interpreted as the objective function of a generative adversarial network (GAN) (Goodfellow et al., 2014) as follows.*

$$\min_{\theta} \mathcal{L}_{SCL} = \max_{\theta} \int_{e} (\tilde{P}_{pos}(e) \log(D(e)) + \tilde{P}_{neg}(e) \log(1 - D(e)))de$$

*where $D(e) = f(\boldsymbol{z}_i, \boldsymbol{z}_j)$ is the discriminator of GAN with edge $e = (v_i, v_j)$ being connected or not, by node representations $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$. Probabilities $\tilde{P}_{pos}(e) = \sigma P_{pos}(e)$ and $\tilde{P}_{neg}(e) = \gamma P_{neg}(e)$ are defined above. (Proof in Appendix C)*

According to Proposition 2.1, the proposed $\mathcal{L}_{\text{SCL}}$ can be interpreted as distinguishing the existence of a certain edge based on the representation vectors of two nodes. Next, we then derive how this design helps, as a regularizer, to alleviate the oversmoothing problem based on positive and negative samples.

**Theorem 2.2.** *The regularization term $\mathcal{L}_{SCL}$ can alleviate the oversmoothing problem (i.e., the node hidden representation vectors are distinguishable based on their connection activities), if the discriminator in Proposition 2.1 reached the optimum, and the optimum is $D^*(e) = \frac{\tilde{P}_{pos}(e)}{\tilde{P}_{pos}(e)+\tilde{P}_{neg}(e)}$. (Proof in Appendix C)*

## 3 Experiments

In this section, we design comprehensive experiments to answer the following research questions.

- RQ1: When do we need more layers of graph neural networks? *(Answered in Section 3.2)*
- RQ2: When it is necessary to be deep, can DrGNN alleviate dimensional collapse and maintain or even boost task performance for deeper architecture? *(Answered in Section 3.3 and Section 3.4)*
- RQ3: Is every component of DrGNN helpful and irreplaceable? *(Answered in Section 3.5)*
- RQ4: In practice, can DrGNN be agnostic to help various off-the-shelf graph neural network architectures? *(Answered in Section 3.6)*

### 3.1 Experiment Setup

**Datasets**. *Cora* (Lu & Getoor, 2003) dataset is a citation network consisting of 5,429 edges and 2,708 scientific publications from 7 classes. The edge in the graph represents the citation of one paper by another. *CiteSeer* (Lu & Getoor, 2003) dataset consists of 3,327 scientific publications which could be categorized into 6 classes, and this citation network has 9,228 edges. *PubMed* (Namata et al., 2012) is a citation network consisting of 88,651 edges and 19,717 scientific publications from 3 classes. *Reddit* (Hamilton et al., 2017b) dataset is extracted from Reddit posts, which consists of 4,584 nodes and 19,460 edges. Notice that we follow the splitting strategy used in (Zhao & Akoglu, 2020) by randomly sampling 3% of the nodes as the training samples, 10% of the nodes as the validation samples, and the remaining 87% as the test samples. Moreover, we follow the OGB benchmark Hu et al. (2020) for the large-scale dataset *OGB-arXiv* (Wang et al., 2020) which is a citation network and consists of 1,166,243 edges and 169,343 nodes from 40 classes. Also, we adopt the non-homophilous benchmark Lim et al. (2021) for the heterophilous version of OGB-arXiv, which is denoted as *arXiv-year* and the edge homophily is just 0.222.

**Baselines**. We compare the performance of our method with the following baselines: (1) GCN (Kipf & Welling, 2017): the vanilla graph convolutional network; (2) GCNII (Chen et al., 2020): an extension of GCN with skip connections and additional identity matrices; (3) DGN (Zhou et al., 2020): the differentiable group normalization for GNNs to normalize nodes within the same group and separate nodes among different groups; (4) PairNorm (Zhao & Akoglu, 2020): a GNN normalization layer designed to prevent node representations from becoming too similar; (5) DropEdge (Rong et al., 2020): a GNN-agnostic framework that randomly removes a certain number of edges from the input graph; (6) RevGCN-Deep (Li et al., 2021): equilibrium model based deep graph neural networks; (7) EGNN (Zhou et al., 2021): dirichlet energy constrained deep graph neural networks; (8) ContraNrom (Guo et al., 2023): a contrastive learning-based layer normalization method. Detailed reproducibility with released code can be found in Appendix D.

### 3.2 When do we need more layers of graph neural networks?

**Case 1: Missing Features**. We first consider a scenario where some attribute values are missing in the input graph. In such cases, shallow GNNs may not perform well because they cannot gather enough information from close neighbors due to the presence of numerous missing values. By increasing the number of layers, GNNs can gather more information and capture latent knowledge to compensate for missing features. To verify this, we conducted the following experiment: we randomly masked $p\%$ of attributes (i.e., setting the masked attributes to be 0), gradually increased the number of layers, and recorded the accuracy for each setting. In this case study, we selected the number of layers from the set $\{2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 40, 50, 60\}$, and the backbone model used was GCN. For a fair comparison, we added residual connection of ResNet (He et al., 2016) to baselines if adding it could enhance the baseline's performance, denoted as (+RC). We repeated the experiments five times and recorded the mean accuracy and standard deviation.

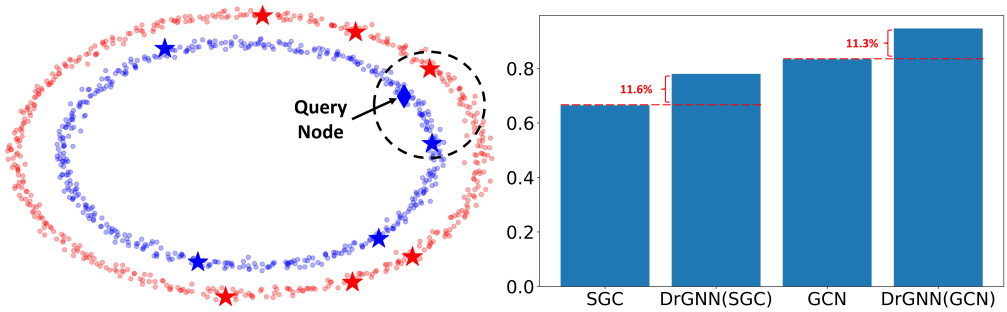
Table 1: Node Classification by Masking $p\%$ of Input Node Attributes ($L$ denotes the number of layers that achieves the best performance).

| Node Feature Missing Rate | | $p = 25\%$ | | $p = 50\%$ | | $p = 75\%$ | |
|---|---|---|---|---|---|---|---|
| Dataset | Method | Accuracy | $L$ | Accuracy | $L$ | Accuracy | $L$ |
| Cora | GCN (+RC) | 0.7503 ± 0.0101 | 7 | 0.7435 ± 0.0048 | 10 | 0.7226 ± 0.0099 | 10 |
| | PairNorm (+RC) | 0.7529 ± 0.0129 | 10 | 0.7482 ± 0.0172 | 20 | 0.7262 ± 0.0178 | 40 |
| | DropEdge (+RC) | 0.7634 ± 0.0112 | 15 | 0.7611 ± 0.0102 | 20 | 0.7297 ± 0.0168 | 8 |
| | GCNII (+RC) | 0.2667 ± 0.0063 | 25 | 0.3351 ± 0.0066 | 25 | 0.2914 ± 0.0106 | 40 |
| | DGN | 0.6850 ± 0.0184 | 30 | 0.6846 ± 0.0147 | 50 | 0.6717 ± 0.0156 | 25 |
| | ContraNorm (+RC) | 0.7319 ± 0.0099 | 2 | 0.7189 ± 0.0091 | 3 | 0.6902 ± 0.0107 | 3 |
| | DrGNN | **0.7915 ± 0.0060** | 10 | **0.7848 ± 0.0043** | 20 | **0.7598 ± 0.0081** | 60 |
| CiteSeer | GCN (+RC) | 0.6141 ± 0.0080 | 4 | 0.5811 ± 0.0093 | 10 | 0.5149 ± 0.0173 | 9 |
| | PairNorm (+RC) | 0.6184 ± 0.0087 | 8 | 0.5947 ± 0.0083 | 20 | 0.5176 ± 0.0075 | 10 |
| | DropEdge (+RC) | 0.6348 ± 0.0156 | 4 | 0.6083 ± 0.0128 | 6 | 0.5240 ± 0.0128 | 10 |
| | GCNII (+RC) | 0.2453 ± 0.0045 | 40 | 0.2338 ± 0.0028 | 20 | 0.2403 ± 0.0046 | 25 |
| | DGN | 0.4560 ± 0.0162 | 20 | 0.4593 ± 0.0117 | 15 | 0.4498 ± 0.0292 | 15 |
| | ContraNorm (+RC) | 0.5893 ± 0.0114 | 2 | 0.5621 ± 0.0111 | 3 | 0.4646 ± 0.0076 | 4 |
| | DrGNN | **0.6524 ± 0.0087** | 20 | **0.6169 ± 0.0063** | 60 | **0.5576 ± 0.0070** | 50 |

From Table 1, we find that when the missing rate is 25%, shallow GCN with residual connections has enough capacity to achieve the best performance. Given that, our proposed method further improves the performance by more than 3.83% on the CiterSeer dataset and 4.08% on the Cora dataset by stacking a few layers. However, when we increase the missing rate to 50% and 75%, we observe that most methods tend to achieve the best performance by stacking more layers. Specifically, PairNorm achieves the best performance at 10 layers when 25% features are missing, while it has the best performance at 40 layers when 75% features are missing. A similar observation could also be found with GCNII on the Cora dataset, DropEdge on the CiteSeer dataset as well as our proposed methods in both datasets. Overall, the experimental results verify that the more features a dataset are missing, the more layers GNNs need to be stacked to achieve better performance. Our explanation for this observation is that if the number of layers increases, more information will be collected from the distant neighbors to recover the missing information of its close neighbors.

**Case 2: Missing Labels**. We then conducted another case study using a toy example to demonstrate that nearby neighbors may not necessarily share similar contents. Initially, we utilized an existing package (specifically, the draw circle function in the Scikit-learn package) to generate a synthetic dataset with 1,000 data points and a noise level set to 0.01. Subsequently, we computed the Euclidean distance between each pair of data points. If the distance between two data points is less than a predefined threshold, we connect them in a graph, resulting in the derivation of the adjacency matrix with added self-loops. Following this, we randomly sampled 1% of the data points as the training set, 9% as the validation set, and 90% as the test set. These data points are visualized in Figure 3a, and the corresponding experimental results are presented in Figure 3b. In Figure 3a, we observed that the query node (i.e., the blue diamond within the dashed circle) could not just rely on its closest labeled neighbor (i.e., the red star within the dashed circle) to predict its label (red or blue) correctly. Exploring longer paths consisting of more similar neighbors can help to accurately predict its label as blue (as indicated by the blue star within the dashed circle). Figure 3b

compares the classification accuracy of shallow and deep GNNs with the same backbones. Notably, deeper GNNs exhibited a significant performance improvement of over 11% compared to shallow ones, contributing to their capability of exploring longer paths in the graph.




(a) (b)

Figure 3: A Toy Example to Demonstrate the Benefit of Deeper GNN Models: (a) Two groups of nodes in the semi-supervised setting. Stars are labeled, dots are unlabeled, and the diamond is the query node. Euclidean distance between two nodes determines the edge connection. (b) Comparison of node classification accuracy between shallow and deeper GNN models using data on the left. The deeper GNNs are realized by DrGNN with the corresponding same backbones.

**Case 3: Heterophilous Graphs**. In this section, we evaluate the performance of DrGNN on a large heterophily graph (i.e., arXiv-year) from the benchmark (Lim et al., 2021). In brief, heterophily means the connected two nodes do not share the same label.

Table 2: Node Classification on the arXiv-year dataset.

| Method | arXiv-year |
|---|---|
| GCN | 0.4602 ± 0.0026 |
| GAT | 0.4605 ± 0.0051 |
| GCNJK | 0.4628 ± 0.0029 |
| GATJK | 0.4580 ± 0.0072 |
| APPNP | 0.3815 ± 0.0026 |
| $H_2$GCN | 0.4909 ± 0.0010 |
| MixHop | **0.5181 ± 0.0017** |
| GPR-GNN | 0.4507 ± 0.0021 |
| GCNII | 0.4721 ± 0.0028 |
| DrGNN (10 layers) | 0.4816 ± 0.0008 |
| DrGNN (20 layers) | 0.4835 ± 0.0008 |
| DrGNN (30 layers) | 0.4871 ± 0.0007 |
| DrGNN (50 layers) | <u>0.4995 ± 0.0006</u> |

As shown in Table 2, our DrGNN achieves the second place performance in terms of node classification accuracy, among other state-of-the-art graph neural network methods. Also, we can observe that stacking more layers can increase the performance of DrGNN, because multi-hop neighbor information is aggregated for the message passing in heterphilous graphs. The reason why DrGNN can not achieve the best is that, although stacking layers can aggregate information from multi-hop neighbors, the loss SCL is still regulating that close neighbors should share similar representations, which is generally based on the homophilous condition for more common settings. Note that our design of DrGNN is not solely for heterphilous graphs but for how to stack layers wisely for upgrading performance when the stacking operation is necessary and unavoidable. Heterophily is not the only reason for stacking graph neural layers; at least, the reasoning can also originate from missing features and labels, as shown above, where we need to stack more graph neural layers to mitigate the missing features by incorporating more neighbors.

### 3.3 Effectiveness Analysis of DrGNN

Motivated by the above subsection, we then aim to answer the question: whether our DrGNN can perform if stacking more layers is inevitable? Therefore, we prepare different deep settings to test if DrGNN can always outperform baseline methods.

In Table 3, for a fair comparison, the backbone model for all methods we used in these experiments is GCN, and we set the dimension of the hidden layer to 50 and vary the number of hidden layers from 2 to 16, 32, and 64 for all methods. Additionally, in Table 4, we test DrGNN on the large-scale dataset OGB-arXiv, where we fix the feature dimension of the hidden layer as 100, set the total iteration to 3000, and choose GCN as the backbone model. Due to memory limitations, we set the number of layers to 2, 10, and 20. The experiments are repeated 5 times, and we record the mean accuracy as well as the standard deviation.

Table 3: Node Classification on Small Datasets with Varying Layers $L$ (GCN as the Backbone).

| Dataset | Method | $L = 2$ | $L = 16$ | $L = 32$ | $L = 64$ |
|---|---|---|---|---|---|
| Cora | GCN | 0.7643 ± 0.0040 | 0.5262 ± 0.0732 | 0.3284 ± 0.0066 | 0.3274 ± 0.0189 |
| | PairNorm | 0.7818 ± 0.0027 | 0.6080 ± 0.0310 | 0.5138 ± 0.0299 | 0.2932 ± 0.0120 |
| | DropEdge | **0.7828 ± 0.0075** | 0.7557 ± 0.0072 | 0.7306 ± 0.0134 | 0.2685 ± 0.0647 |
| | GCNII | 0.6778 ± 0.0065 | 0.7237 ± 0.0055 | 0.7142 ± 0.0015 | 0.7107 ± 0.0047 |
| | DGN | 0.7545 ± 0.0003 | 0.6785 ± 0.0169 | 0.7067 ± 0.0190 | 0.7104 ± 0.0192 |
| | ContraNorm | 0.7682 ± 0.0044 | 0.6590 ± 0.0291 | 0.5128 ± 0.0241 | 0.4328 ± 0.0320 |
| | DrGNN | 0.7768 ± 0.0057 | **0.8002 ± 0.0058** | **0.7961 ± 0.0055** | **0.8022 ± 0.0061** |
| CiteSeer | GCN | 0.6452 ± 0.0072 | 0.4514 ± 0.0987 | 0.2689 ± 0.0099 | 0.2680 ± 0.0093 |
| | PairNorm | 0.6030 ± 0.0153 | 0.2268 ± 0.0398 | 0.2096 ± 0.0029 | 0.2076 ± 0.0033 |
| | DropEdge | 0.6532 ± 0.0068 | 0.6117 ± 0.0229 | 0.5101 ± 0.0430 | 0.2138 ± 0.0198 |
| | GCNII | 0.5912 ± 0.0106 | 0.6180 ± 0.0031 | 0.6159 ± 0.0019 | 0.6101 ± 0.0017 |
| | DGN | 0.4872 ± 0.0168 | 0.4753 ± 0.0591 | 0.4604 ± 0.0162 | 0.4417 ± 0.0219 |
| | ContraNorm | 0.6263 ± 0.0061 | 0.4621 ± 0.0237 | 0.3965 ± 0.0196 | 0.2128 ± 0.0208 |
| | DrGNN | **0.6577 ± 0.0065** | **0.6650 ± 0.0059** | **0.6655 ± 0.0031** | **0.6685 ± 0.0066** |
| PubMed | GCN | 0.7990 ± 0.0017 | 0.5383 ± 0.0200 | 0.5463 ± 0.0391 | 0.5566 ± 0.0086 |
| | PairNorm | 0.8120 ± 0.0076 | 0.4408 ± 0.0683 | 0.3972 ± 0.0094 | 0.3960 ± 0.0097 |
| | DropEdge | 0.8035 ± 0.0020 | 0.7893 ± 0.0042 | 0.7902 ± 0.0032 | 0.3951 ± 0.0108 |
| | GCNII | 0.8070 ± 0.0009 | 0.8094 ± 0.0010 | 0.8089 ± 0.0007 | 0.8097 ± 0.0009 |
| | DGN | 0.7947 ± 0.0358 | 0.7553 ± 0.0295 | 0.7733 ± 0.0143 | 0.7632 ± 0.0226 |
| | ContraNorm | 0.8061 ± 0.0020 | 0.5672 ± 0.0684 | 0.4348 ± 0.0379 | 0.3971 ± 0.0057 |
| | DrGNN | **0.8175 ± 0.0016** | **0.8097 ± 0.0038** | **0.8098 ± 0.0025** | **0.8109 ± 0.0033** |
| Reddit | GCN | 0.8757 ± 0.0054 | 0.8540 ± 0.0451 | 0.3655 ± 0.0251 | 0.3410 ± 0.0288 |
| | PairNorm | 0.7704 ± 0.0052 | 0.8636 ± 0.0448 | 0.6468 ± 0.0429 | 0.1230 ± 0.0299 |
| | DropEdge | 0.8564 ± 0.0059 | 0.8526 ± 0.0046 | 0.5384 ± 0.1049 | 0.1053 ± 0.0148 |
| | GCNII | 0.6184 ± 0.0108 | 0.7157 ± 0.0016 | 0.6972 ± 0.0039 | 0.6963 ± 0.0059 |
| | DGN | 0.7829 ± 0.0137 | 0.7397 ± 0.0371 | 0.6806 ± 0.0639 | 0.5058 ± 0.0754 |
| | ContraNorm | 0.6576 ± 0.0094 | 0.2563 ± 0.0091 | 0.2547 ± 0.0170 | 0.2664 ± 0.0140 |
| | DrGNN | **0.8762 ± 0.0060** | **0.9676 ± 0.0033** | **0.9693 ± 0.0023** | **0.9721 ± 0.0011** |

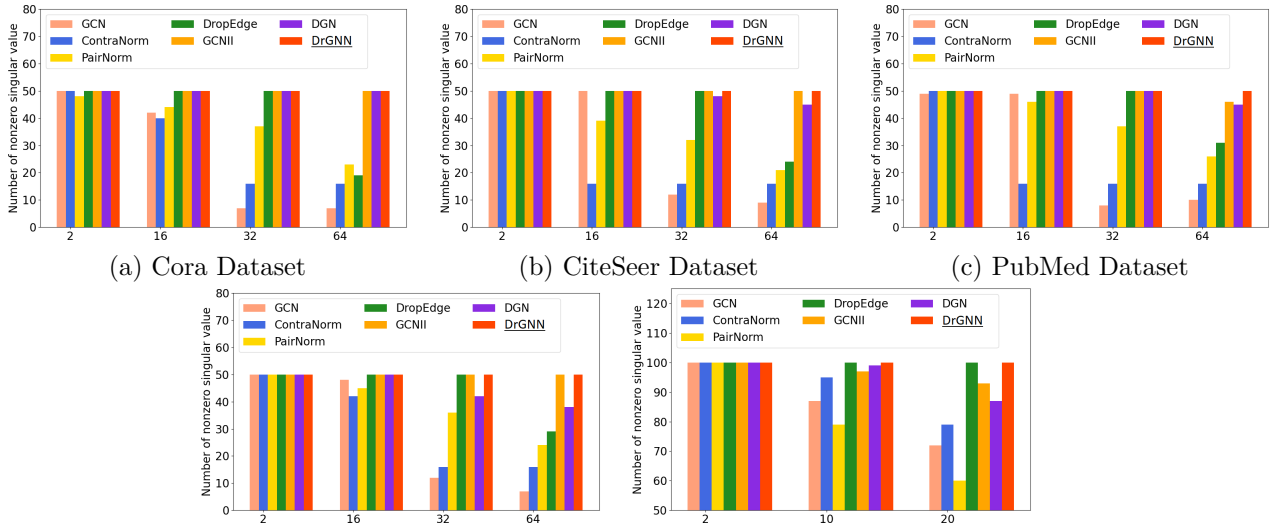
Table 4: Node Classification on Large Dataset with Varying Layers $L$ (GCN as the Backbone).

| Dataset | Method | $L = 2$ | $L = 10$ | $L = 20$ |
|---|---|---|---|---|
| OGB-arXiv | GCN | 0.7136 ± 0.0044 | 0.7021 ± 0.0018 | 0.5377 ± 0.0756 |
| | PairNorm | 0.7186 ± 0.0008 | 0.7158 ± 0.0035 | 0.5796 ± 0.0090 |
| | DropEdge | 0.7178 ± 0.0012 | 0.6531 ± 0.0056 | 0.2198 ± 0.0097 |
| | GCNII | 0.5966 ± 0.0013 | 0.6340 ± 0.0017 | 0.6246 ± 0.0015 |
| | DGN | 0.6039 ± 0.0037 | 0.5746 ± 0.0033 | 0.5027 ± 0.0056 |
| | ContraNorm | 0.7294 ± 0.0025 | 0.6941 ± 0.0030 | 0.5821 ± 0.0324 |
| | DrGNN | **0.7369 ± 0.0014** | **0.7386 ± 0.0006** | **0.7401 ± 0.0009** |

Based on the observation in Table 3 and Table 4, we find that existing SOTA graph de-oversmoothing methods, e.g., DropEdge, PairNorm, ContraNorm, achieve the best performance with the shallow layer (i.e., $L = 2$), and their performance begins to decrease as the number of layers increases. However, DrGNN can outperform robustly even if the GNN becomes deep, and the performance gain is obvious. For example, at layer 16, in

the Cora dataset, DrGNN can achieve 80.02% accuracy. Additional comparison with RevGCN-Deep and EGNN can be found in Appendix E.

To echo with Table 3 and Table 4, we visualize the corresponding number of the nonzero singular values on those datasets in Figure 4. In Figure 4, taking Cora and OGB-arXiv as examples, we observe that (1) PairNorm and ContraNorm begin to suffer from the dimensional collapse issue on both datasets when the number of layers is greater than 10; (2) Dropedge, DGN, and GCNII perform well on the small dataset but fail to preserve the full-rank representation on the large dataset; (3) node representations by DrGNN are full-rank on both datasets, indicating that DrGNN effectively alleviates the dimensional collapse.




(a) Cora Dataset (b) CiteSeer Dataset (c) PubMed Dataset

(d) Reddit Dataset (e) OGB-arXiv Dataset

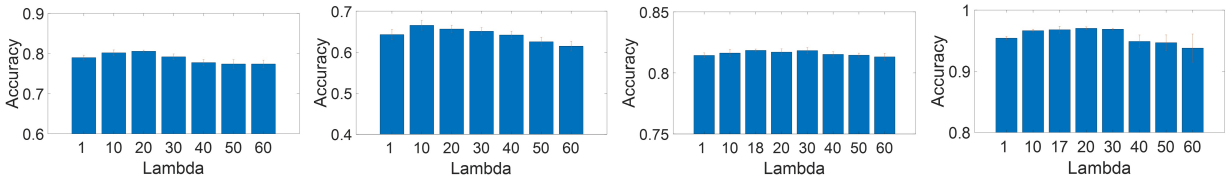
Figure 4: The $x$-axis is the number of layers and the $y$-axis is the number of the non-zero singular values of the covariance matrix of the node representations by different methods.

Although the above results have proven that our DrGNN can outperform robustly in different deep settings, we also observe that there seems to be a peak in terms of the number of graph neural network layers. In other words, after a certain depth, the performance gain seems marginal or missing. For example, in the Cora dataset, the performance of our DrGNN increases from 77.68% to 80.02% from 2 layers to 16 layers. However, the performances are maintained when stacking to the 32nd layer or the 64th layer. Therefore, we may ask **if there are a number of effective neural layers**? and **if the number exists, how can we find it before the training to avoid the heavy hyperparameter tunning process**? Luckily, we found the answer in the following subsection.

### 3.4 Number of Effective Layers in Deep Graph Neural Networks

We conduct the hyperparameter analysis of DrGNN, regarding $\lambda$ in the weight decaying function of Eq. 4. For example, when $\lambda = 10$, the decaying factor for the 10-th layer is 0.3679 (i.e., $e^{-1}$); but for the 30-th layer, it is 0.0049 (i.e., $e^{-3}$). This decay limits the effective information aggregation scope of deeper GNNs because the later stacked layers will become significantly less important. Based on this controlling property of $\lambda$, a natural follow-up question is whether its value depends on the property of input graphs. Basically, it suggests that initially stacking a few layers is effective, but more deeper layers do not help (or do not help much) boost performance.

Interestingly, through our experiments, we find that **the optimal $\lambda$ is very close to the diameter of input graphs (if it is connected) or the largest component (if it does not have many separate components)**. This observation verifies our conjecture regarding the property of $\lambda$ (i.e., it controls the number of effective layers or the number of hops during the message passing aggregation schema of GNNs). Hence, the value of $\lambda$ can be searched around the diameter of the input graph.




(a) Cora (b) CiteSeer (c) PubMed (d) Reddit

Figure 5: Hyperparameter Analysis, i.e., $\lambda$ vs Node Classification Accuracy on Four Datasets.

In details, to analyze the hyperparameter $\lambda$, we fix the feature dimension of the hidden layer to be 50, the total iteration is set to be 3000, the number of layers is set to be 60, the sampling batch size for DrGNN is 10, and GCN is chosen as the backbone model. The experiment is repeated five times for each configuration. In each sub-figure of Figure 5, the $x$-axis is the value of $\lambda$, and the $y$-axis is the accuracy of 60-layer GCN in the above setting.

Specifically, we find that the optimal $\lambda = 20$ on the Cora dataset, the optimal $\lambda = 10$ on the CiteSeer dataset, the optimal $\lambda = 18$ on the PubMed dataset, and the optimal $\lambda = 20$ on the Reddit dataset. Then, natural questions to ask are (1) what determines the optimal value of $\lambda$ in different datasets? (2) can we gather some heuristics to narrow down the hyperparameter search space to efficiently establish effective GNNs?

Thus, we provide our discovery. In Eq. 4, we have analyzed that the decaying factor $\lambda$ controls the number of effective layers in deeper GNNs by introducing the layer-wise dependency. It means that larger $\lambda$ slows down the weight decay and gives considerably large weights to more layers such that they can be effective, and the information aggregation scope of GNN extends as more multi-hop neighbor features are collected and aggregated. In graph theory, diameter represents the scope of the graph, which is the largest value of the shortest path between any node pairs in the graph. Therefore, the optimal $\lambda$ should be restricted by the input graph, i.e., being close to the input graph diameter.

Table 5: Graph Statistics of Each Dataset.

| Metric | Cora | Citeseer | PubMed | Reddit |
|---|---|---|---|---|
| Number of Nodes | 2,708 | 3,327 | 19,717 | 4,854 |
| Connected Graph | No | No | Yes | Yes |
| Number of Components | 78 | 438 | 1 | 1 |
| Diameter of the Graph (or the Largest Component) | 19 | 28 | 18 | 17 |

Interestingly, our experiments reflect this observation. Combining the optimal $\lambda$ in Figure 5 and the diameter in Table 5, for connected graphs PubMed and Reddit, the optimal $\lambda$ is very close to the graph diameter. This also happens to Cora (even though Cora is not connected), because the number of components is not large. As for CiteSeer, the optimal $\lambda$ is less than the diameter of its largest component. A possible reason is that CiteSeer has many (i.e., 438) small components, which shrinks the information propagation scope, such that we do not need to stack many layers and we do not need to enlarge $\lambda$ to the largest diameter (i.e., 28). In general, based on the above analysis, we find the optimal value of $\lambda$ can be searched around the diameter of the input graph.

### 3.5 Ablation Study of DrGNN

Here, we conduct the ablation study to show the effectiveness and irreplaceability of WG-ResNet and SCL in terms of node classification in Table 6. In this experiment, we fix the total iteration set as 3000, and GCN is chosen as the backbone model. For the Cora dataset, the feature dimension of the hidden layer is 50 and the number of layers is 64; for the OGB-arXiv dataset the feature dimension of the hidden layer is 100 and the number of layers is 20. In Table 6, DrGNN-T removes SCL, DrGNN-D removes the weight decaying factor in WG-ResNet and DrGNN-JK replaces the WG-ResNet by Jumping Knowledge (Xu et al., 2018).

In Table 6, we have the following observations (1) comparing DrGNN with DrGNN-T, we find that DrGNN boosts the performance by 1.84% on the Cora dataset after adding SCL, which demonstrates the effectiveness of SCL to alleviate the oversmoothing issue; (2) DrGNN outperforms DrGNN-D on Cora dataset by 5.61%, which shows that DrGNN could alleviate the shading neighbors effect by adding the weight decaying factor;

Table 6: Ablation Study w.r.t. Node Classification Accuracy.

| Method | Cora ($L = 64$) | CiteSeer ($L = 64$) | PubMed ($L = 64$) | Reddit ($L = 64$) | OGB-arXiv ($L = 20$) |
|---|---|---|---|---|---|
| GCN (+RC) | 0.7252 ± 0.0176 | 0.6213 ± 0.0056 | 0.7985 ± 0.0068 | 0.9432 ± 0.0037 | 0.7144 ± 0.0013 |
| DrGNN-D | 0.7498 ± 0.0139 | 0.6567 ± 0.0052 | 0.8050 ± 0.0031 | 0.9654 ± 0.0028 | 0.7363 ± 0.0011 |
| DrGNN-T | 0.7875 ± 0.0092 | 0.5750 ± 0.0244 | 0.8078 ± 0.0047 | 0.9397 ± 0.0042 | 0.7335 ± 0.0024 |
| DrGNN-JK | 0.7955 ± 0.0078 | 0.6600 ± 0.0085 | 0.8061 ± 0.0038 | 0.9659 ± 0.0046 | 0.7368 ± 0.0012 |
| DrGNN | **0.8022 ± 0.0061** | **0.6685 ± 0.0066** | **0.8109 ± 0.0033** | **0.9721 ± 0.0011** | **0.7401 ± 0.0009** |

(3) comparing DrGNN with DrGNN-JK, we verify that our proposed WG-ResNet is more effective than DrGNN-JK. Besides, one drawback of jumping knowledge is its high memory required as the number of layers increases, while our proposed WG-ResNet doesn't; (4) DrGNN outperforms GCN (+RC) by more than 7.7% on the Cora dataset and 2.6% on the OGB-arXiv dataset, which indicates that WG-ResNet could alleviate the shading neighbors effect.

### 3.6 Different Backbones of DrGNN

Here, we show the performance of our proposed DrGNN cooperating with different backbone models (e.g., GAT (Velickovic et al., 2018) and GraphSage (Hamilton et al., 2017a)). In Figure 6, we set the numbers of the hidden layers as 60 for all methods and the dimension of the hidden layer as 50. The total number of training iterations is 1500.

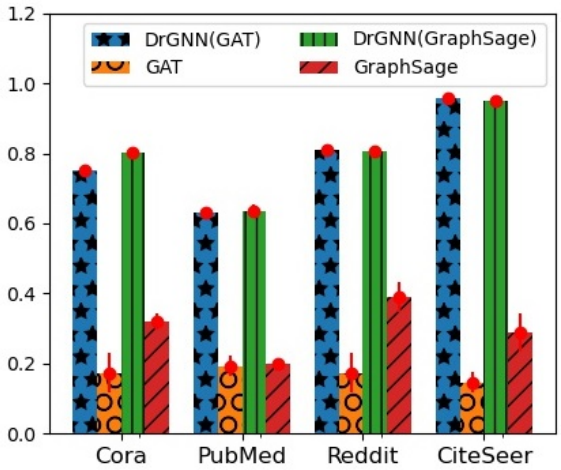



Figure 6: Accuracy of Different Backbone Models with 64 Hidden Layers on Four Datasets.

By observation, we find that both GAT and GraphSage tend to have worse performance when the architecture becomes deeper, and our proposed method DrGNN greatly boosts the performance by 40%-60% on average over four datasets. Specifically, compared with the vanilla GraphSage, our DrGNN boosts its performance by 43% on the CiteSeer dataset and more than 67% on the Reddit dataset.

## 4 Conclusion

In this paper, we focus on building deeper graph neural networks to effectively model graph data and illustrate the oversmoothing cause from the perspective of dimensional collapse. To this end, we first provide insights regarding why the vanilla residual connection of ResNet is not best suited for many deeper graph neural network solutions, i.e., the *shading neighbors* effect. Then, we propose a new residual architecture, Weight-decaying Graph Residual Connection (WG-ResNet), to alleviate this effect. In addition, we propose a Structure-guided Contrastive Learning (SCL) to alleviate the problem from another viewpoint, where we utilize graph topological information, pull the representations of connected node pairs closer, and push remote node pairs farther apart via contrastive learning regularization. Combining WG-ResNet with SCL, an end-to-end model DrGNN is proposed for deep graph neural networks. We provide the theoretical analysis of our proposed method and demonstrate the effectiveness of DrGNN by extensive experiment comparing with state-of-the-art methods.

## References

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, 2020.

John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2011.

S. M. Faisal, Georgios Tziantzioulis, Ali Murat Gok, Nikolaos Hardavellas, Seda Ogrenci Memik, and Srinivasan Parthasarathy. Edge importance identification for energy efficient graph processing. In *2015 IEEE International Conference on Big Data (IEEE BigData 2015), Santa Clara, CA, USA, October 29 - November 1, 2015*, pp. 347–354. IEEE Computer Society, 2015.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014. URL `http://arxiv.org/abs/1406.2661`.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 855–864. ACM, 2016.

Xiaojun Guo, Yifei Wang, Tianqi Du, and Yisen Wang. Contranorm: A contrastive learning perspective on oversmoothing and beyond. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL `https://openreview.net/pdf?id=SM7XkJouWHm`.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017a.

William L. Hamilton, Justine Zhang, Cristian Danescu-Niculescu-Mizil, Dan Jurafsky, and Jure Leskovec. Loyalty in online communities. In *Proceedings of the Eleventh International Conference on Web and Social Media, ICWSM 2017, Montréal, Québec, Canada, May 15-18, 2017*, pp. 540–543. AAAI Press, 2017b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html`.

Tianyu Hua, Wenxiao Wang, Zihui Xue, Sucheng Ren, Yue Wang, and Hang Zhao. On feature decorrelation in self-supervised learning. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pp. 9578–9588. IEEE, 2021.

Kexin Huang and Marinka Zitnik. Graph meta learning via local subgraphs. In *NeurIPS*, 2020.

Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

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

Can M. Le. Edge sampling using local network information. *J. Mach. Learn. Res.*, 22:88:1–88:29, 2021.

Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6437–6449. PMLR, 2021.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34:20887–20902, 2021.

Qing Lu and Lise Getoor. Link-based classification. In Tom Fawcett and Nina Mishra (eds.), *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pp. 496–503. AAAI Press, 2003.

Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, 2012.

Mohammad Norouzi, David J Fleet, and Russ R Salakhutdinov. Hamming distance metric learning. *Advances in neural information processing systems*, 25, 2012.

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

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani (eds.), *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pp. 701–710. ACM, 2014.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *CoRR*, abs/2303.10993, 2023. doi: 10.48550/arXiv.2303.10993. URL `https://doi.org/10.48550/arXiv.2303.10993`.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *CoRR*, abs/1710.10903, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Hongwei Wang and Jure Leskovec. Unifying graph convolutional neural networks and label propagation. *CoRR*, 2020.

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020.

Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9929–9939. PMLR, 2020.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5449–5458. PMLR, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *ICLR*, 2020.

Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. In *NeurIPS*, 2020.

Kaixiong Zhou, Xiao Huang, Daochen Zha, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Dirichlet energy constrained learning for deep graph neural networks. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 21834–21846, 2021.

## A Similarity between Oversmoothing and Dimensional Collapse

In (Rusch et al., 2023), the authors describe *oversmoothing* as a phenomenon that the node representation vectors are indistinguishable from each other and thus deteriorate the performance of downstream tasks. Inspired by this, we could measure the magnitude of graph oversmoothing by the metric of covariance mean as follows:

$$covariance(h) = \frac{1}{n}\sum_i (h_i - \bar{h})(h_i - \bar{h})$$

$$\bar{h} = \frac{1}{n}\sum_i h_i$$

where $h_i$ is the node representation for node $i$ and $covariance(h) = 0$ indicates that the learned representation is indistinguishable and the deep model suffers from an oversmoothing issue. Notice that the dimensional collapse is observed when the covariance matrix of the node representations is not full-rank (i.e., the number of non-zero singular values is less than the dimension of the node representation in Figure 1). When $covariance(h) = 0$, it also indicates that $h_i = h_j = \bar{h}$ for all $i$ and $j$, and the rank of the covariance matrix of the node representation matrix is 0 (While a large value of $covariance(h)$ does not mean that the performance is ). In other words, the graph model suffers from complete collapse, where all node representations shrink to a single point. Thus, we could see that the oversmoothing issue is highly related to dimensional collapse.

## B Visualization of the Weight of Each Layer With Different Weighting Functions

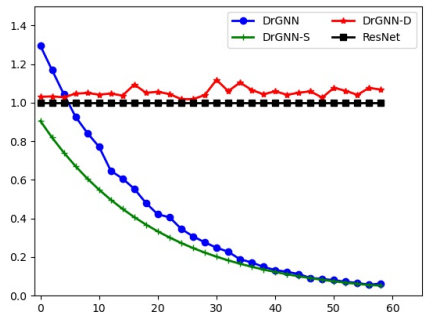



Figure 7: Weight Visualization. The $y$-axis represents the weight of each layer, and the $x$-axis represents the index of each layer, in deeper models.

Here, we visualize the weight of each layer with different weighting functions on the Cora dataset. In this experiment, we fix the feature dimension of the hidden layer to be 50; the total iteration is set to be 3000; the

number of layers is set to be 60; the sampling batch size for DrGNN is 10; GCN is chosen as the backbone model; $\lambda$ is set to be 20. In Figure 7, The $x$-axis is the index of each layer, and the $y$-axis is the weight for each layer. DrGNN-S removes the similarity measurement $e^{cos(\boldsymbol{H}^{(1)},\tilde{\boldsymbol{H}}^{(l)})}$ in Eq. 5 and DrGNN-D removes the decaying weight factor and only keeps the exponential cosine similarity $e^{cos(\boldsymbol{H}^{(1)},\tilde{\boldsymbol{H}}^{(l)})}$ to measure the weight for each layer. DrGNN-S achieves the simplified WG-ResNet in DrGNN, which removes the exponential cosine similarity $e^{cos(\boldsymbol{H}^{(1)},\tilde{\boldsymbol{H}}^{(l)})}$ in DrGNN. By observation, we find that (1) ResNet sets the weight of each layer to be 1, which easily leads to *shading neighbors* effect when stacking more layers, because the faraway neighbor information becomes more dominant in the GCN information aggregation; (2) without weight decaying factor, the weight for each layer in DrGNN-D fluctuates because they are randomly independent. More specially, the weights for the last several layers (e.g., L=58 or L=60) are larger than the weights for the first several layers, which contradicts the intuition that the first several layers should be more important than the last several layers; (3) the weights for each layer in both DrGNN and DrGNN-S reduce as the number of layers increase, which suggests that both of them could address the *shading neighbors* effect to some extents; (4) combining the results from Table 6, DrGNN achieves better performance than DrGNN-S, as it imposes larger weights to the first several layers, which verifies that the learnable similarity $sim(\boldsymbol{H}^{(1)},\tilde{\boldsymbol{H}}^{(l)})$ achieves better performance with the enlarged hypothesis space for neural networks.

## C Theoritical Proof

**Proposition 2.1.** $\mathcal{L}_{\text{SCL}}$, the loss function based on contrastive learning, can be interpreted as the objective function of a generative adversarial network (GAN) (Goodfellow et al., 2014) as follows.

$$\min_{\theta} \mathcal{L}_{\text{SCL}} = \max_{\theta} \int_{e} (\tilde{P}_{pos}(e)\log(D(e)) + \tilde{P}_{neg}(e)\log(1-D(e)))de$$

where $D(e) = f(\boldsymbol{z}_i, \boldsymbol{z}_j)$ is the discriminator of GAN with edge $e = (v_i, v_j)$ being connected or not, by node representations $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$. Probabilities $\tilde{P}_{pos}(e_{ij}) = \sigma_{ij}P_{pos}(e_{ij})$ and $\tilde{P}_{neg}(e_{ik}) = \gamma_{ik}P_{neg}(e_{ik})$.

*Proof.* Represent $D(e)$ by $f(\boldsymbol{z}_i, \boldsymbol{z}_j)$, then we have

$$\begin{aligned}
\min_{\theta} \mathcal{L}_{\text{SCL}} &= -\mathbb{E}_{v_i \in V}[\mathbb{E}_{v_j \in \mathcal{N}_i}(\sigma_{ij}\log(f(\boldsymbol{z}_i, \boldsymbol{z}_j))) + \mathbb{E}_{v_k \in \bar{\mathcal{N}}_i}(\gamma_{ik}\log(1-f(\boldsymbol{z}_i, \boldsymbol{z}_k)))] \\
&= -\min_{\theta} \int_{e} (P_{pos}(e)\sigma)\log(D(e)) + (P_{neg}(e)\gamma)\log(1-D(e))de \\
&= \max_{\theta} \int_{e} (\tilde{P}_{pos}(e)\log(D(e)) + \tilde{P}_{neg}(e)\log(1-D(e)))de
\end{aligned}$$

□

**Theorem 2.2.** The regularization term $\mathcal{L}_{\text{SCL}}$ can alleviate the oversmoothing problem (i.e., the node hidden representation vectors are distinguishable based on their connection activities), if the discriminator in Proposition 2.1 reached the optimum, and the optimum is $D^*(e) = \frac{\tilde{P}_{pos}(e)}{\tilde{P}_{pos}(e)+\tilde{P}_{neg}(e)}$.

*Proof.* Based on Proposition 2.1, a free $D(e)$ is replaced by $D(e) = f(\boldsymbol{z}_i, \boldsymbol{z}_j)$.

According to Eq 5, $f(\boldsymbol{a}, \boldsymbol{b}) = e^{\frac{\boldsymbol{a}\boldsymbol{b}^\top}{||\boldsymbol{a}||||\boldsymbol{b}||}}$, then, $f(\boldsymbol{a}, \boldsymbol{b}) \in [e^{-1}, e^{1}]$.

Thus, suppose that $\frac{a}{a+b} \geq e^{-1}$, it is easy to prove that the function $alog(x) + blog(e-x)$ has the maximum in $[e^{-1}, e^{1}]$ at $\frac{a}{a+b}$, where $a$ and $b$ are positive constant, $e$ is Euler's number, and $x$ is a variable.

Hence, we first have $D^*(e) = \frac{\tilde{P}_{pos}(e)}{\tilde{P}_{pos}(e)+\tilde{P}_{neg}(e)}$.

Given $D^*(e) = \frac{\tilde{P}_{pos}(e)}{\tilde{P}_{pos}(e)+\tilde{P}_{neg}(e)}$, and $\tilde{P}_{pos}(e_{ij}) = \sigma_{ij}P_{pos}(e_{ij})$ and $\tilde{P}_{neg}(e_{ik}) = \gamma_{ik}P_{neg}(e_{ik})$ defined in Proposition 2.1, we have

$$\begin{aligned} D^*(e) &= \frac{\sigma * P_{pos}(e)}{\sigma * P_{pos}(e) + \gamma * P_{neg}(e)} \\ &= \frac{P(e|y=1)P(y=1)}{P(e|y=1)P(y=1) + P(e|y=0)P(y=0)} \\ &= P(y=1|e) \end{aligned}$$

Therefore, $D(e)$ is towards maximizing the conditional log-likelihood $P(y=1|e)$, where $y=1$ indicates edge $e$ is a positive (i.e., exiting) edge.

To sum up, the discriminator $D(e)$ is defined as the similarity measurement of a node pair in terms of their representation vectors, as stated in Proposition 2.1. In other words, if $D(e)$ is able to distinguish whether a node pair is a negative pair or not, the hidden representations of these two nodes (negative pair) are then distinguishable. Thus, *we can conclude that when $\mathcal{L}_{SCL}$ achieves the optimal solution, the model could successfully discriminate the difference of the hidden representation vectors*, and alleviating the oversmoothing issue. In practice, the optimum is usually approximated by the model convergence. □

## D Reproducibility

For a fair comparison, we set the dropout rate to 0.5, the weight decay rate to 0.0005, and the total number of iterations to 1500 for all baseline methods in Table 1 and Table 3; if not specialized, GCN is chosen as the backbone, and the dimension of each layer is set to 50 for all the graph neural network baseline methods.

In Section 3.5, for DrGNN and DrGNN-S, we sample 10 instances and 5 neighbors for each class from the training set, $dist(\cdot)$ is the hamming distance, and $f(\cdot)$ is the cosine similarity measurement.

The experiments are repeated 10 times if not otherwise specified. All of the real-world datasets are publicly available. The experiments are performed on a Windows machine with a 16GB RTX 5000 GPU.

The code of our algorithm is in an anonymous link [1]. We provide the detailed experimental setting for each experiment shown in Table 7.

Table 7: Hyperparameters for DrGNN shown in Table 3.

| Method | DrGNN |
|---|---|
| Cora | $\lambda = 20, \alpha = 0.03$ |
| CiteSeer | $\lambda = 10, \alpha = 0.02$ |
| PubMed | $\lambda = 18, \alpha = 0.1$ |
| Reddit | $\lambda = 20, \alpha = 0.02$ |

Moreover, we set the learning rate to be 0.001 and the optimizer is RMSProp, which is one variant of ADAGRAD (Duchi et al., 2011).

## E Additional Effectiveness Analysis

We conduct the additional experiments by comparing our proposed method with RevGCN-Deep (Li et al., 2021) and EGNN (Zhou et al., 2021). We set the number of layers for all baseline methods to 60 for Cora, Citeseer, PubMed, and Reddit. For the OGB-arXiv dataset, we set the number of layers to 10 for all methods.

[1] https://drive.google.com/file/d/1cbNI74lhTb3LsOKhgVHT1btNz20ZLb60/view?usp=sharing

Table 8: Additional Node Classification Comparison.

| Method | Cora ($L = 60$) | CiteSeer ($L = 60$) | PubMed ($L = 60$) | Reddit ($L = 60$) | OGB-arXiv ($L = 10$) |
|---|---|---|---|---|---|
| RevGCN-Deep | 0.7458 ± 0.0084 | 0.5137 ± 0.0099 | 0.8139 ± 0.0015 | 0.8853 ± 0.0383 | 0.7354 ± 0.0009 |
| EGNN | 0.7961 ± 0.0036 | 0.6566 ± 0.0060 | 0.8138 ± 0.0026 | 0.8772 ± 0.0040 | 0.7247 ± 0.0015 |
| GearGNN | 0.8059 ± 0.0028 | 0.6655 ± 0.0117 | 0.8185 ± 0.0016 | 0.9721 ± 0.0011 | 0.7401 ± 0.0009 |

## F Additional Ablation Study of DrGNN

In this section, we conduct an additional ablation study to evaluate the performance of each component (i.e., the cosine similarity function in Eq. 4 and SCL formulated in Eq. 5). In Table 9, DrGNN-S denotes the variant of DrGNN after removing the cosine similarity function in Eq. 4 and SCL denotes the variant of DrGNN by removing the WG-ResNet. We have the following observations: (1). DrGNN outperforms DrGNN-S on most datasets except PubMed, which suggests that introducing the layer similarity (i.e., cosine similarity between the first layer and the $l$-th layer) can increase the performance of DrGNN(2). Compared to ContraNorm, our proposed structure-guided contrastive learning (SCL) can further boost the performance by more than 29% on average, which demonstrates the effectiveness of SCL over ContraNorm.

Table 9: Additional Ablation Study w.r.t. Node Classification Accuracy.

| Method | Cora ($L = 64$) | CiteSeer ($L = 64$) | PubMed ($L = 64$) | Reddit ($L = 64$) | OGB-arXiv ($L = 20$) |
|---|---|---|---|---|---|
| DrGNN | **0.8022 ± 0.0061** | **0.6685 ± 0.0066** | 0.8109 ± 0.0033 | **0.9721 ± 0.0011** | **0.7401 ± 0.0009** |
| DrGNN-S | 0.7931 ± 0.0153 | 0.6555 ± 0.0085 | **0.8173 ± 0.0030** | 0.9621 ± 0.0035 | 0.7335 ± 0.0024 |
| ContraNorm | 0.4328 ± 0.0320 | 0.2128 ± 0.0208 | 0.3971 ± 0.0057 | 0.2664 ± 0.0140 | 0.5821 ± 0.0324 |
| SCL | 0.5751 ± 0.0264 | 0.3933 ± 0.0076 | 0.7601 ± 0.0043 | 0.9305 ± 0.0266 | 0.6952 ± 0.0011 |

## G Additional Hyperarameter Analysis

Here, we conduct additional hyperparameter analysis of DrGNN, i.e., $\alpha$ in the overall loss function of Eq 1.

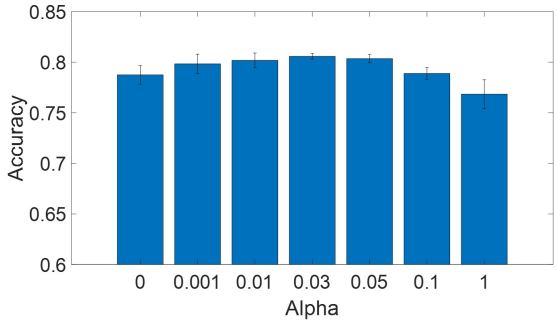



Figure 8: Hyperparameter Analysis, i.e., $\alpha$ vs Node Classification Accuracy.

To analyze the hyperparameter $\alpha$ in DrGNN, we fix the feature dimension of the hidden layer to be 50, the total iteration is set to be 3000, the number of layers is set to be 60, the sampling batch size for DrGNN is 10, GCN is chosen as the backbone model, and the dataset is Cora. We gradually increase the value of $\alpha$ and record the accuracy. The experiment is repeated five times in each setting. In Figure 8, the x-axis is $\alpha$ and the y-axis is the accuracy score. By observation, when $\alpha = 1$, the performance is worst and the performance begins to increase by decreasing the value of $\alpha$. It achieves the best accuracy when $\alpha = 0.03$. The performance starts to decrease again if we further decrease the value of $\alpha$. We conjecture that when $\alpha$ is large, it will dominate the overall objective function, thus jeopardizing the classification performance. Besides, the performance also decreases if we set the value of $\alpha$ to be a small number (i.e., $\alpha = 0.001$). In addition, comparing with the performance without using SCL regularization (i.e., $\alpha = 0$), our proposed

method with $\alpha = 0.03$ can boost the performance by more than 1.8%, which demonstrates that our proposed SCL alleviates the issue of oversmoothing to some extent.

## H Efficiency Analysis

Here, we conduct an efficiency analysis regarding our proposed method in the Cora dataset. We fix the feature dimension of the hidden layer to be 50, the total iteration is set to be 1500, the sampling batch size for DrGNN and DrGNN-S is 10, and GCN is chosen as the backbone model. We gradually increase the number of layers and record the running time.

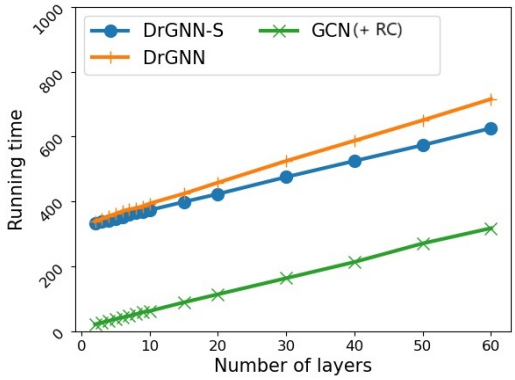



Figure 9: The Number of Layers vs Running Time (in seconds) on Cora.

In Figure 9, the $x$-axis is the number of layers and the $y$-axis is the running time in seconds. We observe that the running time of both DrGNN and DrGNN-S is linearly proportional to the number of layers. Comparing the running time of DrGNN, the running time of DrGNN-S is further reduced after the weighting function in DrGNN (e.g., $\text{sim}(\cdot)$) is replaced by a constant.

## I Sampling Method for SCL

To realize SCL expressed in Eq. 5, we need to get the positive nodes $v_j$ and negative nodes $v_k$ towards the selected central node $v_i$. To avoid iterating over all existing nodes or randomly sampling several nodes, we propose to sample positive nodes $v_j$ and negative nodes $v_k$ from the star subgraph $S_i$ of the central node $v_i$. Moreover, to make the sampling scalable and to reduce the search space of negative nodes, we propose a batch sampling method.

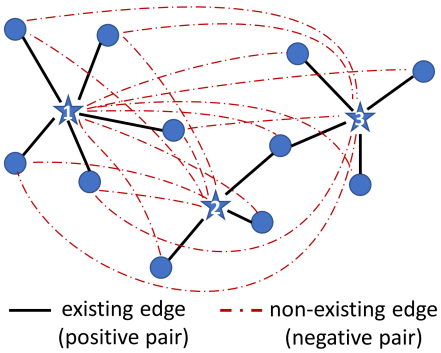



Figure 10: Batch Sampling. Each star node in the figure corresponds to node $v_i$ in Eq. 5.

As shown in Figure 10, the batch size is controlled by the number of central nodes (i.e., star nodes in the figure). For each central node, the positive nodes are those 1-hop neighbors, and the negative nodes consist of unreachable nodes. In our batch sampling, we strictly constrain that the positive nodes are only from the 1-hop neighborhood for the following three reasons: (1) they are efficient to be accessed; (2) considering all $k$-hop neighbors as positive will enlarge the scope of positive nodes and further decrease the intimacy of the directly connected nodes; (3) 1-hop positive nodes in the star subgraph can preserve enough useful information, compared to the positive nodes from the whole graph. For the third point, we prove it through the *graph influence loss* (Huang & Zitnik, 2020) in Proposition I.1, and the formal definition of *graph influence loss* is given in the following paragraph after Proposition I.1.

**Proposition I.1** (Bounded Graph Influence Loss for Sampling Positive Pairs Locally)**.** *Taking GCN as an example of GNN, the graph influence loss $R(v_c)$ on node $v_c$ w.r.t* ***positive nodes from the whole graph*** *against* ***positive nodes from the 1-hop neighborhood star subgraph*** *is bounded by $R(v_c) \leq (n-d_c)\frac{\mu}{(D_{GM}^{\bar{\mathcal{P}}_*})^{|\bar{\mathcal{P}}_*|}}$ , where $n$ is the number of nodes, $d_c$ is the degree of node $v_c$ including the self-loop, $\mu$ is a constant, $\bar{\mathcal{P}}_*$ is the path from center node $v_c$ to a 1-hop outside node $v_s$ which has the maximal node influence $I_{v_c,v_s}$, and $|\bar{\mathcal{P}}_*|$ denotes the number of nodes in path $\bar{\mathcal{P}}_*$.*

*Proof.* According to the assumption of (Wang & Leskovec, 2020), $\sigma(\cdot)$ can be identity function and $\boldsymbol{W}^{(\cdot)}$ can be identity matrix. Then, the hidden node representation (of node $v_c$) in the last layer of GCN can be written as follows.

$$\boldsymbol{h}_c^{(\infty)} = \frac{1}{D_{c,c}} \sum_{v_i \in \mathcal{N}_c} A_{c,i} \boldsymbol{h}_i^{(\infty)}$$

Then, based on the above equation, we can iteratively replace $\boldsymbol{h}_i^{(\infty)}$ with its neighbors until the representation $\boldsymbol{h}_s^{(\infty)}$ of node $v_s$ is included. The extension procedure is written as follows.

$$\begin{aligned}\boldsymbol{h}_c^{(\infty)} = & \frac{1}{D_{c,c}} \sum_{v_i \in \mathcal{N}_c} A_{c,i} \frac{1}{D_{i,i}} \sum_{v_j \in \mathcal{N}_i} A_{i,j} \ \ldots \\ & \frac{1}{D_{k,k}} \sum_{v_s \in \mathcal{N}_k} A_{k,s} \boldsymbol{h}_s^{(\infty)}\end{aligned}$$

The above equation suggests that the influence from the positive node $v_s$ to the center node $v_c$ is through the path $\mathcal{P} = (v_c, v_i, v_j, \ldots, v_k, v_s)$.

Following the above path formation and assuming the edge weight $A(i,j)$ as the positive constant, according to (Huang & Zitnik, 2020), we can obtain the node influence $I_{v_c,v_s}$ of $v_s$ on $v_c$ as follows.

$$I_{v_c,v_s} = \|\partial \boldsymbol{h}_c^{(\infty)} / \partial \boldsymbol{h}_s^{(\infty)}\| \leq \frac{\mu}{(D_{GM}^{\bar{\mathcal{P}}})^{|\bar{\mathcal{P}}|}}$$

where $\mu$ is a constant, $D_{GM}^{\bar{\mathcal{P}}}$ is the geometric mean of the degree of nodes sitting in path $\bar{\mathcal{P}}$, and $\bar{\mathcal{P}}$ is the path from the positive node $v_s$ to the center node $v_c$ that could generate the maximal multiplication of normalized edge weight, $|\bar{\mathcal{P}}|$ denotes the number of nodes in path $\bar{\mathcal{P}}$.

The above analysis suggests that the node influence of positive long-distance nodes is decaying.

Hence, the graph influence loss about learning node $v_c$ from **the whole graph positive nodes** versus **from the 1-hop localized positive nodes** can be expressed as follows.

$$\begin{aligned}I_G(v_c) - I_L(v_c) &= I_{v_c,v_1} + I_{v_c,v_2} + \ldots + I_{v_c,v_{n-d_c}} \\ &\leq \sum_{i=1}^{n-d_c} \frac{\mu_i}{(D_{GM}^{\bar{\mathcal{P}}_i})^{|\bar{\mathcal{P}}_i|}} \\ &\leq (n-d_c) \frac{\mu_*}{(D_{GM}^{\bar{\mathcal{P}}_*})^{|\bar{\mathcal{P}}_*|}}\end{aligned}$$

where $I_G(v_c)$ denotes global influence, $I_L(v_c)$ is the influence for star subgraph, $d_c$ is the degree of node $v_c$ (including self-loop), and $\frac{\mu_*}{(D_{GM}^{\bar{\mathcal{P}}_*})^{|\bar{\mathcal{P}}_*|}}$ is the maximal among all $\frac{\mu_i}{(D_{GM}^{\bar{\mathcal{P}}_i})^{|\bar{\mathcal{P}}_i|}}$. □

Specifically, the graph influence loss (Huang & Zitnik, 2020) $R(v_c)$ can be expressed as $R(v_c) = I_G(v_c) - I_L(v_c)$, which is determined by the global graph influence on $v_c$ (i.e., $I_G(v_c)$) and the star subgraph influence on $v_c$ (i.e., $I_L(v_c)$). Then, to compute the graph influence $I_G(v_c)$, we need to compute the node influence of each node $v_j$ to node $v_c$, where node $v_j$ is reachable from node $v_c$. Based on the final output node representation vectors, the node influence is expressed as $I_{v_c,v_j} = \|\partial \boldsymbol{h}_c^{(\infty)} / \partial \boldsymbol{h}_j^{(\infty)}\|$, and the norm can be any subordinate norm (Wang & Leskovec, 2020). Then, $I_G(v_c)$ is computed by the $L1$-norm of the following vector, i.e., $I_G(v_c) = \|[I_{v_c,v_1}, I_{v_c,v_2}, \ldots, I_{v_c,v_n}]\|_1$. Similarly, we can compute the star subgraph influence $I_L(v_c)$ on node $v_c$. The only difference is that we collect each reachable node $v_j$ in the star subgraph $L$ (i.e., 1-hop neighbors of $v_c$). Overall, in Proposition I.1, we show why positive pairs can be locally sampled with the support from graph influence loss of a node representation vector output by the GCN final layer.