Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings

Yu Chen
Rensselaer Polytechnic Institute
cheny39@rpi.edu

Lingfei Wu*
IBM Research
lwu@email.wm.edu

Mohammed J. Zaki
Rensselaer Polytechnic Institute
zaki@cs.rpi.edu

Abstract

In this paper, we propose an end-to-end graph learning framework, namely **I**terative **D**eep **G**raph **L**earning (IDGL), for jointly and iteratively learning graph structure and graph embedding. The key rationale of IDGL is to learn a better graph structure based on better node embeddings, and vice versa (i.e., better node embeddings based on a better graph structure). Our iterative method dynamically stops when the learned graph structure approaches close enough to the graph optimized for the downstream prediction task. In addition, we cast the graph learning problem as a similarity metric learning problem and leverage adaptive graph regularization for controlling the quality of the learned graph. Finally, combining the anchor-based approximation technique, we further propose a scalable version of IDGL, namely IDGL-ANCH, which significantly reduces the time and space complexity of IDGL without compromising the performance. Our extensive experiments on nine benchmarks show that our proposed IDGL models can consistently outperform or match the state-of-the-art baselines. Furthermore, IDGL can be more robust to adversarial graphs and cope with both transductive and inductive learning.

1 Introduction

Recent years have seen a significantly growing amount of interest in graph neural networks (GNNs), especially on efforts devoted to developing more effective GNNs for node classification [29, 36, 17, 52], graph classification [60, 43] and graph generation [47, 37, 61]. Despite GNNs' powerful ability in learning expressive node embeddings, unfortunately, they can only be used when graph-structured data is available. Many real-world applications naturally admit network-structured data (e.g., social networks). However, these intrinsic graph-structures are not always optimal for the downstream tasks. This is partially because the raw graphs were constructed from the original feature space, which may not reflect the "true" graph topology after feature extraction and transformation. Another potential reason is that real-world graphs are often noisy or even incomplete due to the inevitably error-prone data measurement or collection. Furthermore, many applications such as those in natural language processing [7, 57, 58] may only have sequential data or even just the original feature matrix, requiring additional graph construction from the original data matrix.

To address these limitations, we propose an end-to-end graph learning framework, namely **I**terative **D**eep **G**raph **L**earning (IDGL), for jointly and iteratively learning the graph structure and the GNN parameters that are optimized toward the downstream prediction task. The key rationale of our IDGL

*Corresponding author.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

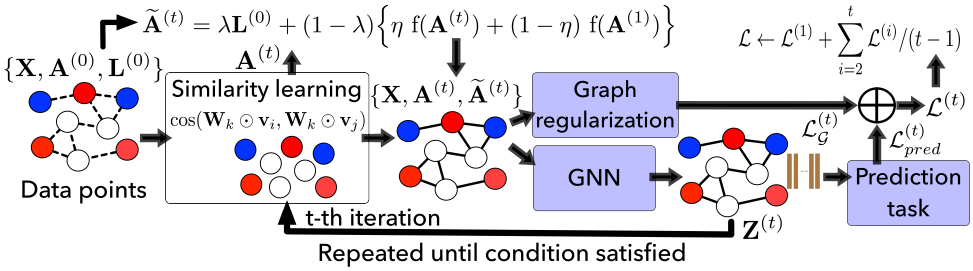


Figure 1: Overall architecture of the proposed IDGL framework. Dashed lines (in data points on left) indicate the initial noisy graph topology $\mathbf{A}^{(0)}$ (if not available we use a kNN graph).

framework is to learn a better graph structure based on better node embeddings, and at the same time, to learn better node embeddings based on a better graph structure. In particular, IDGL is a novel iterative method that aims to search for an implicit graph structure that augments the initial graph structure (if not available we use a kNN graph) with the goal of optimizing the graph for downstream prediction tasks. The iterative method adjusts when to stop in each mini-batch when the learned graph structure approaches close enough to the graph optimized for the downstream task.

Furthermore, we present a graph learning neural network that uses multi-head self-attention with epsilon-neighborhood sparsification for constructing a graph. Moreover, unlike the work in [25] that directly optimizes an adjacency matrix without considering the downstream task, we learn a graph metric learning function by optimizing a joint loss combining both task-specific prediction loss and graph regularization loss. Finally, we further propose a scalable version of our IDGL framework, namely IDGL-ANCH, by combining the anchor-based approximation technique, which reduces the time and memory complexity from quadratic to linear with respect to the numbers of graph nodes.

In short, we summarize the main contributions as follows:

- We propose a novel end-to-end graph learning framework (IDGL) for jointly and iteratively learning the graph structure and graph embedding. IDGL dynamically stops when the learned graph structure approaches the optimized graph (for prediction). To the best of our knowledge, we are the first to introduce iterative learning for graph structure learning.
- Combining the anchor-based approximation technique, we further propose a scalable version of IDGL, namely IDGL-ANCH, which achieves linear complexity in both computational time and memory consumption with respect to the number of graph nodes.
- Experimental results show that our models consistently outperform or match the state-of-the-art baselines on various downstream tasks. More importantly, IDGL can be more robust to adversarial graph examples and can cope with both transductive and inductive learning.

2 Iterative Deep Graph Learning Framework

2.1 Problem Formulation

Let the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be represented as a set of n nodes $v_i \in \mathcal{V}$ with an initial node feature matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$, edges $(v_i, v_j) \in \mathcal{E}$ (binary or weighted) formulating an initial noisy adjacency matrix $\mathbf{A}^{(0)} \in \mathbb{R}^{n \times n}$, and a degree matrix $\mathbf{D}_{ii}^{(0)} = \sum_j \mathbf{A}_{ij}^{(0)}$. Given a noisy graph input $\mathcal{G} := \{\mathbf{A}^{(0)}, \mathbf{X}\}$ or only a feature matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$, the deep graph learning problem we consider in this paper is to produce an optimized graph $\mathcal{G}^* := \{\mathbf{A}^{(*)}, \mathbf{X}\}$ and its corresponding graph node embeddings $\mathbf{Z} = f(\mathcal{G}^*, \theta) \in \mathbb{R}^{h \times n}$, with respect to some (semi-)supervised downstream task. It is worth noting that we assume that the graph noise is only from graph topology (the adjacency matrix) and the node feature matrix $\mathbf{X}$ is noiseless. The more challenging scenario where both graph topology and node feature matrix are noisy, is part of our future work. Without losing the generality, in this paper, we consider both node-level and graph-level prediction tasks.

2.2 Graph Learning and Graph Embedding: A Unified Perspective

Graph topology is crucial for a GNN to learn expressive graph node embeddings. Most of existing GNN methods simply assume that the input graph topology is perfect, which is not necessarily true

in practice since real-world graphs are often noisy or incomplete. More importantly, the provided input graph(s) may not be ideal for the supervised downstream tasks since most of raw graphs are constructed from the original feature space which may fail to reflect the "true" graph topology after high-level feature transformations. Some previous works [52] mitigate this issue by reweighting the importance of neighborhood node embeddings using self-attention on previously learned node embeddings, which still assumes that the original graph connectivity information is noiseless.

To handle potentially noisy input graph, we propose our novel IDGL framework that formulates the problem as an iterative learning problem which jointly learns the graph structure and the GNN parameters. The key rationale of our IDGL framework is to learn a better graph structure based on better node embeddings, and in the meanwhile, to learn better node embeddings based on a better graph structure, as shown in Fig. 2. Unlike most existing methods that construct graphs based on raw node features, the node embeddings learned by GNNs (optimized toward the downstream task) could provide useful information for learning better graph structures. On the other hand, the newly learned graph structures could be a better graph input for GNNs to learn better node embeddings.

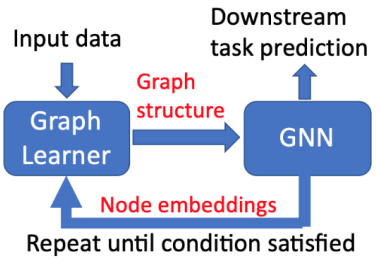


Figure 2: A sketch of the proposed IDGL framework.

In particular, IDGL is a novel iterative method that aims to search for an implicit graph structure that augments the initial graph structure (if not available we use a kNN graph) for downstream prediction tasks. The iterative method dynamically stops in each mini-batch when the learned graph structure approaches close enough to the optimized graph (with respect to the downstream task) based on our proposed stopping criterion. Moreover, the process of graph construction can be optimized toward the downstream task in an end-to-end manner.

2.3 Graph Learning as Similarity Metric Learning

Previous methods (e.g., [15]) that model the graph learning problem as learning a joint discrete probability distribution on the edges of the graph have shown promising performance. However, since they optimize the edge connectivities by assuming that the graph nodes are known, they are unable to cope with the inductive setting (with new nodes during testing). To overcome this issue, we cast the graph structure learning problem as similarity metric learning, which will be jointly trained with the prediction model dedicated to a downstream task.

Graph similarity metric learning. Common options for metric learning include cosine similarity [44, 54], radial basis function (RBF) kernel [59, 34] and attention mechanisms [51, 23]. A good similarity metric function is supposed to be learnable and expressively powerful. Although our framework is agnostic to various similarity metric functions, without loss of generality, we design a weighted cosine similarity as our metric function, $s_{ij} = \cos(\mathbf{w} \odot \mathbf{v}_i, \mathbf{w} \odot \mathbf{v}_j)$, where $\odot$ denotes the Hadamard product, and $\mathbf{w}$ is a learnable weight vector which has the same dimension as the input vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, and learns to highlight different dimensions of the vectors. Note that the two input vectors could be either raw node features or computed node embeddings.

To stabilize the learning process and increase the expressive power, we extend our similarity metric function to a multi-head version (similar to the observations in [51, 52]). Specifically, we use m weight vectors (each one representing one perspective) to compute m independent similarity matrices using the above similarity function and take their average as the final similarity:

$$s_{ij}^p = \cos(\mathbf{w}_p \odot \mathbf{v}_i, \mathbf{w}_p \odot \mathbf{v}_j), \quad s_{ij} = \frac{1}{m} \sum_{p=1}^{m} s_{ij}^p \tag{1}$$

Intuitively, s_{ij}^p computes the cosine similarity between the two input vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, for the p-th perspective, where each perspective considers one part of the semantics captured in the vectors.

Graph sparsification via ε-neighborhood. Typically an adjacency matrix (computed from a metric) is supposed to be non-negative but s_{ij} ranges between $[-1, 1]$. In addition, many underlying graph structures are much more sparse than a fully connected graph which is not only computationally expensive but also might introduce noise (i.e., unimportant edges). We hence proceed to extract a symmetric sparse non-negative adjacency matrix $\mathbf{A}$ from $\mathbf{S}$ by considering only the ε-neighborhood

for each node. Specifically, we mask off (i.e., set to zero) those elements in $\mathbf{S}$ which are smaller than a non-negative threshold ε.

Anchor-based scalable metric learning. The above similarity metric function like Eq. (1) computes similarity scores for all pairs of graph nodes, which requires $\mathcal{O}(n^2)$ complexity for both computational time and memory consumption, rendering significant scalablity issue for large graphs. To address the scalability issue, inspired by previous anchor-based methods [41, 55], we design an anchor-based scalable metric learning technique which learns a node-anchor affinity matrix $\mathbf{R} \in \mathbb{R}^{n \times s}$ (i.e., requires $\mathcal{O}(ns)$ for both time and space complexity where s is the number of anchors) between the node set $\mathcal{V}$ and the anchor set $\mathcal{U}$. Note that s is a hyperparameter which is tuned on the development set.

Specifically, we randomly sample a set of $s \in \mathcal{U}$ anchors from the node set $\mathcal{V}$, where s is usually much smaller than n in large graphs. The anchor embeddings are thus set to the corresponding node embeddings. Therefore, Eq. (1) can be rewritten as the following:

$$a_{ik}^p = \cos(\mathbf{w}_p \odot \mathbf{v}_i, \mathbf{w}_p \odot \mathbf{u}_k), \quad a_{ik} = \frac{1}{m} \sum_{p=1}^{m} a_{ik}^p \tag{2}$$

where a_{ik} is the affinity score between node v_i and anchor u_k. Similarly, we apply the ε-neighborhood sparsification technique to the node-anchor affinity scores a_{ik} to obtain a sparse and non-negative node-anchor affinity matrix $\mathbf{R}$.

2.4 Graph Node Embeddings and Prediction

Although the initial graph could be noisy, it typically still carries rich and useful information regarding true graph topology. Ideally, the learned graph structure $\mathbf{A}$ could be supplementary to the original graph topology $\mathbf{A}^{(0)}$ to formulate an optimized graph for GNNs with respect to the downstream task. Therefore, with the mild assumption that the optimized graph structure is potentially a "shift" from the initial graph structure, we combine the learned graph with the initial graph,

$$\widetilde{\mathbf{A}}^{(t)} = \lambda \mathbf{L}^{(0)} + (1-\lambda)\Big\{\eta\, \mathrm{f}(\mathbf{A}^{(t)}) + (1-\eta)\, \mathrm{f}(\mathbf{A}^{(1)})\Big\} \tag{3}$$

where $\mathbf{L}^{(0)} = {\mathbf{D}^{(0)}}^{-1/2} \mathbf{A}^{(0)} {\mathbf{D}^{(0)}}^{-1/2}$ is the normalized adjacency matrix of the initial graph. $\mathbf{A}^{(t)}$ and $\mathbf{A}^{(1)}$ are the two adjacency matrices computed at the t-th and 1-st iterations (using Eq. (1)), respectively. The adjacency matrix is further row normalized, namely, $\mathrm{f}(\mathbf{A})_{ij} = A_{ij} / \sum_j A_{ij}$.

Note that $\mathbf{A}^{(1)}$ is computed from the raw node features $\mathbf{X}$, whereas $\mathbf{A}^{(t)}$ is computed from the previously updated node embeddings $\mathbf{Z}^{(t-1)}$ that is optimized toward the downstream prediction task. Therefore, we make the final learned graph structure as their linear combination weighted by a hyperparameter η, so as to combine the advantages of both. Finally, another hyperparameter λ is used to balance the trade-off between the learned graph structure and the initial graph structure. If such an initial graph structure is not available, we instead use a kNN graph constructed based on raw node features $\mathbf{X}$ using cosine similarity.

Our graph learning framework is agnostic to various GNN architectures (that take as input a node feature matrix and an adjacency matrix to compute node embeddings) and prediction tasks. In this paper, we adopt a two-layered GCN [29] where the first layer (denoted as GNN_1) maps the raw node features $\mathbf{X}$ to the intermediate embedding space, and the second layer (denoted as GNN_2) further maps the intermediate node embeddings $\mathbf{Z}$ to the output space.

$$\mathbf{Z} = \mathrm{ReLU}(\mathrm{MP}(\mathbf{X}, \widetilde{\mathbf{A}})\mathbf{W}_1), \;\; \widehat{\mathbf{y}} = \sigma(\mathrm{MP}(\mathbf{Z}, \widetilde{\mathbf{A}})\mathbf{W}_2), \;\; \mathcal{L}_{\text{pred}} = \ell(\widehat{\mathbf{y}}, \mathbf{y}) \tag{4}$$

where $\sigma(\cdot)$ and $\ell(\cdot)$ are task-dependent output function and loss function, respectively. For instance, for a classification task, $\sigma(\cdot)$ is a softmax function for predicting a probability distribution over a set of classes, and $\ell(\cdot)$ is a cross-entropy function for computing the prediction loss. $\mathrm{MP}(\cdot, \cdot)$ is a message passing function, and in GCN, $\mathrm{MP}(\mathbf{F}, \widetilde{\mathbf{A}}) = \widetilde{\mathbf{A}}\mathbf{F}$ for a feature/embedding matrix $\mathbf{F}$ and normalized adjacency matrix $\widetilde{\mathbf{A}}$ which we obtain using Eq. (3).

Node-anchor message passing. Note that a node-anchor affinity matrix $\mathbf{R}$ serves as a weighted adjacency matrix of a bipartite graph $\mathcal{B}$ allowing only direct connections between nodes and anchors. If we regard a direct travel between a node and an anchor as one-step transition described by $\mathbf{R}$, built

upon theories of stationary Markov random walks [42], we can actually recover both the node graph $\mathcal{G}$ and the anchor graph $\mathcal{Q}$ from $\mathbf{R}$ by computing the two-step transition probabilities. Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ denote a row-normalized adjacency matrix for the node graph $\mathcal{G}$, and $A_{ij} = p^{(2)}(v_j|v_i)$ indicate the two-step transition probability from node v_i to node v_j, $\mathbf{A}$ can be recovered from $\mathbf{R}$,

$$\mathbf{A} = \mathbf{\Delta}^{-1}\mathbf{R}\mathbf{\Lambda}^{-1}\mathbf{R}^\top \tag{5}$$

where $\Lambda \in \mathbb{R}^{s\times s}$ ($\Lambda_{kk} = \sum_{i=1}^{n} R_{ik}$) and $\Delta \in \mathbb{R}^{n\times n}$ ($\Delta_{ii} = \sum_{k=1}^{s} R_{ik}$) are both diagonal matrices. Similarly, we can recover the row-normalized adjacency matrix $\mathbf{B} \in \mathbb{R}^{s\times s}$ for the anchor graph $\mathcal{Q}$,

$$\mathbf{B} = \mathbf{\Lambda}^{-1}\mathbf{R}^\top\mathbf{\Delta}^{-1}\mathbf{R} \tag{6}$$

A detailed proof of recovering node and anchor graphs from the affinity matrix is provided in Appendix A.1. While explicitly computing a node adjacency matrix $\mathbf{A}$ from $\mathbf{R}$ (Eq. (5)) and directly performing message passing over the node graph $\mathcal{G}$ (Eq. (4)) are expensive in both time complexity ($\mathcal{O}(n^2 s)$) and space complexity ($\mathcal{O}(n^2)$), one can instead equivalently decompose the above process (denoted as MP_{12}) into two steps: i) node-to-anchor message passing MP_1 and ii) anchor-to-node message passing MP_2, over the node-anchor bipartite graph $\mathcal{B}$, formulated as follows,

$$\text{MP}_{12}(\mathbf{F}, \mathbf{R}) = \text{MP}_2(\mathbf{F}', \mathbf{R}), \quad \mathbf{F}' = \text{MP}_1(\mathbf{F}, \mathbf{R}) \tag{7}$$

where $\text{MP}_1(\mathbf{F}, \mathbf{R}) = \mathbf{\Lambda}^{-1}\mathbf{R}^\top\mathbf{F}$ aims to pass message $\mathbf{F}$ from the nodes $\mathcal{V}$ to the anchors $\mathcal{U}$, and $\text{MP}_2(\mathbf{F}', \mathbf{R}) = \mathbf{\Delta}^{-1}\mathbf{R}\mathbf{F}'$ aims to further pass the message $\mathbf{F}'$ aggregated on the anchors back to the nodes. Finally, we can obtain $\text{MP}_{12}(\mathbf{F}, \mathbf{R}) = \mathbf{\Delta}^{-1}\mathbf{R}\mathbf{\Lambda}^{-1}\mathbf{R}^\top\mathbf{F} = \mathbf{A}\mathbf{F}$ where $\mathbf{A}$ is the node adjacency matrix recovered from $\mathbf{R}$ using Eq. (5). In this way, we reduce both time and space complexity to $\mathcal{O}(ns)$. Therefore, we can rewrite the regular node embedding and prediction equations defined in Eqs. (3) and (4) as follows,

$$\mathbf{Z} = \text{ReLU}(\text{MP}_a(\mathbf{X}, \{\mathbf{L}^{(0)}, \mathbf{R}^{(t)}, \mathbf{R}^{(1)}\})\mathbf{W}_1), \;\; \widehat{\mathbf{y}} = \sigma(\text{MP}_a(\mathbf{Z}, \{\mathbf{L}^{(0)}, \mathbf{R}^{(t)}, \mathbf{R}^{(1)}\})\mathbf{W}_2) \tag{8}$$

where $\text{MP}_a(\cdot,\cdot)$ is a hybrid message passing function with the same spirit of Eq. (3), defined as,

$$\text{MP}_a(\mathbf{F}, \{\mathbf{L}^{(0)}, \mathbf{R}^{(t)}, \mathbf{R}^{(1)}\}) = \lambda\text{MP}(\mathbf{F}, \mathbf{L}^{(0)}) + (1-\lambda)\Big\{\eta\text{MP}_{12}(\mathbf{F}, \mathbf{R}^{(t)}) + (1-\eta)\text{MP}_{12}(\mathbf{F}, \mathbf{R}^{(1)})\Big\} \tag{9}$$

Note that we use the same $\text{MP}(\cdot,\cdot)$ function defined in Eq. (4) for performing message passing over $\mathbf{L}^{(0)}$ which is typically sparse in practice, and $\mathbf{F}$ can either be $\mathbf{X}$ or $\mathbf{Z}$.

2.5 Graph Regularization

Although combining the learned graph $\mathbf{A}^{(t)}$ with the initial graph $\mathbf{A}^{(0)}$ is an effective way to approach the optimaized graph, the quality of the learned graph $\mathbf{A}^{(t)}$ plays an important role in improving the quality of the final graph $\widetilde{\mathbf{A}}^{(t)}$. In practice, it is important to control the smoothness, connectivity and sparsity of the resulting learned graph $\mathbf{A}^{(t)}$, which faithfully reflects the graph topology with respect to the initial node attributes $\mathbf{X}$ and the downstream task.

Let each column of the feature matrix $\mathbf{X}$ be considered as a graph signal. A widely adopted assumption for graph signals is that values change smoothly across adjacent nodes. Given an undirected graph with a symmetric weighted adjacency matrix A, the smoothness of a set of n graph signals $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ is usually measured by the Dirichlet energy [2],

$$\Omega(\mathbf{A}, \mathbf{X}) = \frac{1}{2n^2}\sum_{i,j} A_{ij}||\mathbf{x}_i - \mathbf{x}_j||^2 = \frac{1}{n^2}\text{tr}(\mathbf{X}^T\mathbf{L}\mathbf{X}) \tag{10}$$

where $\text{tr}(\cdot)$ denotes the trace of a matrix, $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the graph Laplacian, and $\mathbf{D} = \sum_j A_{ij}$ is the degree matrix. As can be seen, minimizing $\Omega(\mathbf{A}, \mathbf{X})$ forces adjacent nodes to have similar features, thus enforcing smoothness of the graph signals on the graph associated with $\mathbf{A}$.

However, solely minimizing the smoothness loss will result in the trivial solution $\mathbf{A} = 0$. Also, it is desirable to have control of how sparse the resulting graph is. Following [25], we impose additional constraints on the learned graph,

$$f(\mathbf{A}) = \frac{-\beta}{n}\mathbf{1}^T\log(\mathbf{A}\mathbf{1}) + \frac{\gamma}{n^2}||\mathbf{A}||_F^2 \tag{11}$$

Algorithm 1 General Framework for IDGL and IDGL-ANCH

Input: $\mathbf{X}$, $\mathbf{y}[, \mathbf{A}^{(0)}]$
Parameters: m, ε, α, β, γ, λ, δ, T, η, $k[, s]$
Output: Θ, $\hat{\mathbf{y}}$, $\tilde{\mathbf{A}}^{(t)}$ or $\mathbf{R}^{(t)}$
$[\mathbf{A}^{(0)} \leftarrow \text{kNN}(\mathbf{X}, k)]$ {kNN-graph if no initial $\mathbf{A}^{(0)}$}
$t \leftarrow 1$
StopCond $\leftarrow |\mathbf{A}^{(t)} - \mathbf{A}^{(t-1)}|_F^2 > \delta |\mathbf{A}^{(1)}|_F^2$ **if** IDGL **else** $|\mathbf{R}^{(t)} - \mathbf{R}^{(t-1)}|_F^2 > \delta |\mathbf{R}^{(t)}|_F^2$
while (($t == 1$ **or** StopCond) **and** $t \leqslant T$ **do**
  if IDGL **then**
    $\mathbf{A}^{(t)} \leftarrow \text{GL}(\mathbf{X})$ **or** $\text{GL}(\mathbf{Z}^{(t-1)})$ using Eq. (1) {Refine adj. matrix}
    $\tilde{\mathbf{A}}^{(t)} \leftarrow \{\mathbf{A}^{(0)}, \mathbf{A}^{(t)}, \mathbf{A}^{(1)}\}$ using Eq. (3) {Combine refined and raw adj. matrices}
    $\mathbf{Z}^{(t)} \leftarrow \text{GNN}_1(\tilde{\mathbf{A}}^{(t)}, \mathbf{X})$ using Eq. (4) {Refine node embeddings}
  else
    $\mathbf{R}^{(t)} \leftarrow \text{GL}(\mathbf{X}, \mathbf{X}_{\mathcal{U}})$ **or** $\text{GL}(\mathbf{Z}^{(t-1)}, \mathbf{Z}_{\mathcal{U}}^{(t-1)})$ using Eq. (2) {Refine affinity matrix}
    $\mathbf{Z}^{(t)} \leftarrow \text{GNN}_1(\{\mathbf{A}^{(0)}, \mathbf{R}^{(t)}, \mathbf{R}^{(1)}\}, \mathbf{X})$ using Eqs. (8) and (9) {Refine node embeddings}
  end if
  $\hat{\mathbf{y}} \leftarrow \text{GNN}_2(\tilde{\mathbf{A}}^{(t)}, \mathbf{Z}^{(t)})$ using Eq. (4) **if** IDGL **else** $\text{GNN}_2(\{\mathbf{A}^{(0)}, \mathbf{R}^{(t)}, \mathbf{R}^{(1)}\}, \mathbf{Z}^{(t)})$ using Eqs. (8) and (9)
  $\mathcal{L}_{\text{pred}}^{(t)} \leftarrow \text{LOSS}_1(\hat{\mathbf{y}}, \mathbf{y})$ using Eq. (4)
  $\mathcal{L}_{\mathcal{G}}^{(t)} \leftarrow \alpha\Omega(\mathbf{A}^{(t)}, \mathbf{X}) + f(\mathbf{A}^{(t)})$ **if** IDGL **else** $\alpha\Omega(\hat{\mathbf{B}}^{(t)}, \mathbf{X}^{\mathcal{U}}) + f(\hat{\mathbf{B}}^{(t)})$ **where** $\hat{\mathbf{B}}^{(t)} = \mathbf{R}^{(t)\top} \mathbf{\Delta}^{-1} \mathbf{R}^{(t)}$
  $\mathcal{L}^{(t)} \leftarrow \mathcal{L}_{\text{pred}}^{(t)} + \mathcal{L}_{\mathcal{G}}^{(t)}$ **and** $t \leftarrow t + 1$
end while
$\mathcal{L} \leftarrow \mathcal{L}^{(1)} + \sum_{i=2}^{t} \mathcal{L}^{(i)}/(t-1)$
Back-propagate $\mathcal{L}$ to update model weights Θ {In training phase only}

where $||\cdot||_F$ denotes the Frobenius norm of a matrix. The first term penalizes the formation of disconnected graphs via the logarithmic barrier, and the second term controls sparsity by penalizing large degrees due to the first term.

We then define the overall graph regularization loss as the sum of the above losses $\mathcal{L}_{\mathcal{G}} = \alpha\Omega(\mathbf{A}, \mathbf{X}) + f(\mathbf{A})$, which is able to control the smoothness, connectivity and sparsity of the learned graph where α, β and γ are all non-negative hyperparameters.

Anchor graph regularization. As shown in Eq. (6), we can obtain a row-normalized adjacency matrix $\mathbf{B}$ for the anchor graph $\mathcal{Q}$ in $\mathcal{O}(ns^2)$ time complexity. In order to control the quality of the learned node-anchor affinity matrix $\mathbf{R}$ (which can result in implicit control of the quality of the node adjacency matrix $\mathbf{A}$), we apply the aforementioned graph regularization techniques to the anchor graph. It is worthing noting that our proposed graph regularization loss is only applicable to non-negative and symmetric adjacency matrices [26]. Therefore, instead of applying graph regularization to $\mathbf{B}$ which is often not symmetric, we opt to apply graph regularization to its unnormalized version $\hat{\mathbf{B}} = \mathbf{R}^\top \mathbf{\Delta}^{-1} \mathbf{R}$ as $\mathcal{L}_{\mathcal{G}} = \alpha\Omega(\hat{\mathbf{B}}, \mathbf{X}^{\mathcal{U}}) + f(\hat{\mathbf{B}})$, where $\mathbf{X}^{\mathcal{U}}$ denotes the set of anchor embeddings sampled from the set of node embeddings $\mathbf{X}$.

2.6 Joint Learning with A Hybrid Loss

Compared to previous works which directly optimize the adjacency matrix based on either graph regularization loss [26], or task-dependent prediction loss [15], we propose to jointly and iteratively learning the graph structure and the GNN parameters by minimizing a hybrid loss function combining both the task prediction loss and the graph regularization loss, namely, $\mathcal{L} = \mathcal{L}_{\text{pred}} + \mathcal{L}_{\mathcal{G}}$.

The full algorithm of the IDGL framework is presented in Algorithm 1. As we can see, our model repeatedly refines the adjacency matrix with updated node embeddings (Eq. (1)), and refines the node embeddings (Eqs. (3) and (4)) with the updated adjacency matrix until the difference between adjacency matrices at consecutive iterations are smaller than certain threshold. Note that compared to using a fixed number of iterations globally, our dynamic stopping criterion is more beneficial, especially for mini-batch training. At each iteration, a hybrid loss combining both the task-dependent prediction loss and the graph regularization loss is computed. After all iterations, the overall loss is back-propagated through all previous iterations to update the model parameters. Notably, Algorithm 1

is also applicable to IDGL-ANCH. The major differences between IDGL and IDGL-ANCH are how we compute adjacency (or affinity) matrix, and perform message passing and graph regularization.

3 Experiments

In this section, we conduct extensive experiments to verify the effectiveness of IDGL and IDGL-ANCH in various settings. The implementation of our proposed models is publicly available at `https://github.com/hugochan/IDGL`.

Datasets and baselines. The benchmarks used in our experiments include four citation network datasets (i.e., Cora, Citeseer, Pubmed and ogbn-arxiv) [48, 21] where the graph topology is available, three non-graph datasets (i.e., Wine, Breast Cancer (Cancer) and Digits) [11] where the graph topology does not exist, and two text benchmarks (i.e., 20Newsgroups data (20News) and movie review data (MRD)) [32, 46] where we treat a document as a graph containing each word as a node. The first seven datasets are all for node classification tasks in the transductive setting, and we follow the experimental setup of previous works [29, 15, 21]. The later two datasets are for graph-level prediction tasks in the inductive setting. Please refer to Appendix C.1 for detailed data statistics.

Our main baseline is LDS [15] which however is incapable of handling inductive learning problems, we hence only report its results on transductive datasets. In addition, for citation network datasets, we include other GNN variants (i.e., GCN [29], GAT [52], GraphSAGE [18], APPNP [30], H-GCN [20] and GDC [31]) as baselines. For non-graph and text benchmarks where the graph topology is unavailable, we conceive various GNN_{kNN} baselines (i.e., GCN_{kNN}, GAT_{kNN} and $\text{GraphSAGE}_{\text{kNN}}$) where a kNN graph on the data set is constructed during preprocessing before applying a GNN model. For text benchmarks, we include a BiLSTM [19] baseline. The reported results are averaged over 5 runs with different random seeds.

Experimental results. Table 1 shows the results of transductive experiments. First of all, IDGL outperforms all baselines in 4 out of 5 benchmarks. Besides, compared to IDGL, IDGL-ANCH is more scalable and can achieve comparable or even better results. In a scenario where the graph structure is available, compared to the state-of-the-art GNNs and graph learning models, our models achieve either significantly better or competitive results, even though the underlying GNN component of our models is a vanilla GCN. When the graph topology is not available (thus GNNs are not directly applicable), compared to graph learning baselines, IDGL consistently achieves much better results on all datasets. Compared to our main graph learning baseline LDS, our models not only achieve significantly better performance, but also are more scalable. The results of inductive experiments are shown in Table 2. Unlike LDS which cannot handle inductive setting, the good performance on 20News and MRD demonstrates the capability of IDGL on inductive learning.

Table 1: Summary of results in terms of classification accuracies (in percent) on transductive benchmarks. The star symbol indicates that we ran the experiments. The dash symbol indicates that reported results were unavailable or we were not able to run the experiments due to memory issue.

Model	Cora	Citeseer	Pubmed	ogbn-arxiv	Wine	Cancer	Digits
GCN	81.5	70.3	79.0	71.7 (0.3)	—	—	—
GAT	83.0 (0.7)	72.5 (0.7)	79.0 (0.3)	—	—	—	—
GraphSAGE	77.4 (1.0)	67.0 (1.0)	76.6 (0.8)	71.5 (0.3)	—	—	—
APPNP	—	**75.7 (0.3)**	79.7 (0.3)	—	—	—	—
H-GCN	**84.5 (0.5)**	72.8 (0.5)	79.8 (0.4)	—	—	—	—
GCN+GDC	83.6 (0.2)	73.4 (0.3)	78.7 (0.4)	—	—	—	—
LDS	84.1 (0.4)	75.0 (0.4)	—	—	97.3 (0.4)	94.4 (1.9)	92.5 (0.7)
GCN_{kNN}*	—	—	—	—	95.9 (0.9)	94.7 (1.2)	89.5 (1.3)
GAT_{kNN}*	—	—	—	—	95.8 (3.1)	88.6 (2.7)	89.8 (0.6)
$\text{GraphSAGE}_{\text{kNN}}$*	—	—	—	—	96.5 (1.1)	92.8 (1.0)	88.4 (1.8)
LDS*	83.9 (0.6)	74.8 (0.3)	—	—	96.9 (1.4)	93.4 (2.4)	90.8 (2.5)
IDGL	**84.5 (0.3)**	74.1 (0.2)	—	—	97.8 (0.6)	**95.1 (1.0)**	93.1 (0.5)
IDGL-ANCH	84.4 (0.2)	72.0 (1.0)	**83.0 (0.2)**	**72.0 (0.3)**	**98.1 (1.1)**	94.8 (1.4)	**93.2 (0.9)**

Ablation study. Table 3 shows the ablation study results on different modules in our models. we can see a significant performance drop consistently for both IDGL and IDGL-ANCH on all datasets by turning off the iterative learning component (i.e., iterating only once), indicating its effectiveness. Besides, we can see the benefits of jointly training the model with the graph regularization loss.

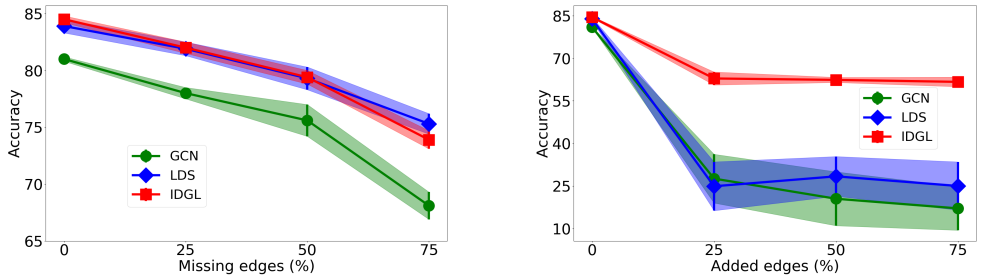
Table 2: Summary of results in terms of classification accuracies or regression scores (R^2) (in percent) on inductive benchmarks.

Methods	20News	MRD
BiLSTM	80.0 (0.4)	53.1 (1.4)
GCN_{kNN}	81.3 (0.6)	60.1 (1.5)
IDGL	**83.6 (0.4)**	**63.7 (1.8)**
IDGL-ANCH	82.9 (0.3)	62.9 (0.4)

Table 3: Ablation study on various node/graph classification datasets.

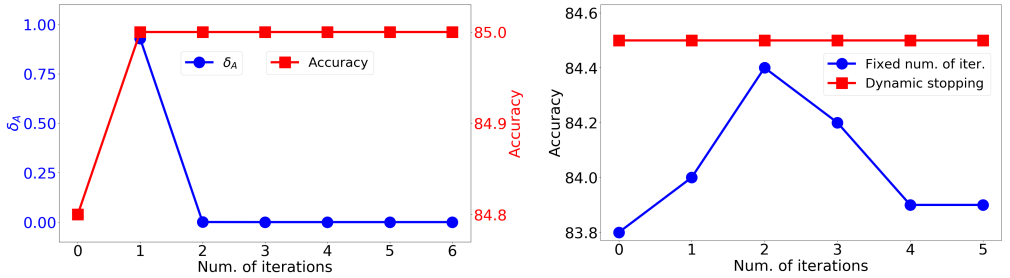
Methods	Cora	Citeseer	Wine	Cancer	Digits	20News
IDGL	**84.5 (0.3)**	**74.1 (0.2)**	**97.8 (0.6)**	**95.1 (1.0)**	**93.1 (0.5)**	**83.6 (0.4)**
w/o graph reg.	84.3 (0.4)	71.5 (0.9)	97.3 (0.8)	94.9 (1.0)	91.5 (0.9)	83.4 (0.5)
w/o IL	83.5 (0.6)	71.0 (0.8)	97.2 (0.8)	94.7 (0.9)	92.4 (0.4)	83.0 (0.4)
IDGL-ANCH	**84.4 (0.2)**	**72.0 (1.0)**	**98.1 (1.1)**	**94.8 (1.4)**	**93.2 (0.9)**	**82.9 (0.3)**
w/o graph reg.	83.2 (0.8)	70.1 (0.8)	97.4 (1.8)	**94.8 (1.4)**	92.0 (1.3)	82.5 (0.7)
w/o IL	83.6 (0.2)	68.6 (0.7)	96.4 (1.5)	94.0 (2.6)	93.0 (0.4)	82.3 (0.3)

Model analysis. To evaluate the robustness of IDGL to adversarial graphs, we construct graphs with random edge deletions or additions. Specifically, for each pair of nodes in the original graph, we randomly remove (if an edge exists) or add (if no such edge) an edge with a probability 25%, 50% or 75%. As shown in Fig. 3, compared to GCN and LDS, IDGL achieves better or comparable results in both scenarios. While both GCN and LDS completely fail in the edge addition scenario, IDGL performs reasonably well. We conjecture this is because the edge addition scenario is more challenging than the edge deletion scenario by incorporating misleading additive random noise to the initial graph. And Eq. (3) is formulated as a form of skip-connection, by lowering the value of λ (i.e., tuned on the development set), we enforce the model to rely less on the initial noisy graph.



(a) Edge deletion (b) Edge addition

Figure 3: Test accuracy ($\pm$ standard deviation) in percent for the edge attack scenarios on Cora.



(a) Convergence plot on Cora. (b) Stopping strategy comparison on Cora.

Figure 4: Convergence and stopping strategy study on Cora (Single run results).

In Fig. 4a (and Appendix B.1), we show the evolution of the learned adjacency matrix and accuracy through iterations in the iterative learning procedure in the testing phase. We compute the difference between adjacency matrices at consecutive iterations as $\delta_A^{(t)} = ||\mathbf{A}^{(t)} - \mathbf{A}^{(t-1)}||_F^2 / ||\mathbf{A}^{(t)}||_F^2$ which typically ranges from 0 to 1. As we can see, both the adjacency matrix and accuracy converge quickly. This empirically verifies the analysis we made on the convergence property of IDGL in Appendix A.2. Please note that this convergence property is not due to the oversmoothing effect of GNNs [56, 33], because we only employ a two-layered GCN as the underlying GNN module of IDGL in our experiments.

We compare the training efficiency of IDGL and IDGL-ANCH with other baselines. As shown in Table 4, IDGL is consistently faster than LDS, but in general, they are comparable. Note that IDGL has comparable model size compared to LDS. For instance, on the Cora data, the number of trainable parameters of IDGL is 28,836, and for LDS, it is 23,040. And we see a large speedup of IDGL-ANCH compared to IDGL. Note that we were not able to run IDGL on Pubmed because of memory limitation. The theoretical complexity analysis is provided in Appendix A.3.

We also empirically study the stopping strategy (Fig. 4b and Appendix B.2), visualize the graph structures learned by IDGL (Appendix B.3), and conduct hyperparameter analysis (Appendix B.4). Details on model settings are provided in Appendix C.2.

4 Related Work

The problem of graph structure learning has been widely studied in different fields from different perspectives. In the field of graph signal processing, researchers have explored various ways of learning graphs from data [10, 12, 53, 27, 3, 1], with certain structural constraints (e.g., sparsity) on the graphs. This problem has also been studied in the literature of clustering analysis [4, 22] where they aimed to simultaneously perform the clustering task and learn similarity relationships among objects. These works all focused on unsupervised learning setting without considering any supervised downstream tasks, and were incapable of handling inductive learning problems. Other related works include structure inference in probabilistic graphical models [9, 66, 62], and graph generation [38, 49], which have a different goal from ours.

Table 4: Mean and standard deviation of training time (5 runs) on various benchmarks (in seconds).

Data	Cora	Citeseer	Pubmed
GCN	**3 (1)**	**5 (1)**	**29 (4)**
GAT	26 (5)	28 (5)	—
LDS	390 (82)	585 (181)	—
IDGL	237 (21)	563 (100)	—
w/o IL	49 (8)	61 (15)	—
IDGL-ANCH	83 (6)	261 (50)	323 (53)
w/o IL	28 (4)	69 (9)	71 (17)

In the field of GNNs [29, 16, 18, 35, 63], there is a line of research on developing robust GNNs [50] that are invulnerable to adversarial graphs by leveraging attention-based methods [5], Bayesian methods [13, 64], graph diffusion-based methods [31], and various assumptions on graphs (e.g., low rank and sparsity) [14, 24, 65]. These methods usually assume that the initial graph structure is available. Recently, researchers have explored methods to automatically construct a graph of objects [45, 8, 34, 15, 40] or words [39, 6, 7] when applying GNNs to non-graph structured data. However, these methods merely optimize the graphs toward the downstream tasks without the explicit control of the quality of the learned graphs. More recently, [15] proposed the LDS model for jointly learning the graph and the parameters of GNNs by leveraging the bilevel optimization technique. However, by design, their method is unable to handle the inductive setting. Our work is also related to Transformer-like approaches [51] that learn relationships among objects by leveraging multi-head attention mechanism. However, these methods do not focus on the graph learning problem and were not designed to utilize the initial graph structure.

5 Conclusion

We proposed a novel IDGL framework for jointly and iteratively learning the graph structure and embeddings optimized for the downstream task. Experimental results demonstrate the effectiveness and efficiency of the proposed models. In the future, we plan to explore effective techniques for handling more challenging scenarios where both graph topology and node features are noisy.

Broader Impact

The fundamental goal of our research is to develop a method for jointly learning graph structures and embeddings that are optimized for (semi-)supervised downstream tasks. Our technique can be widely applied to a large range of applications, including social network analysis, natural language processing (e.g., question answering and text generation), drug discovery and community detection. Conceptually, any application with the purpose of jointly learning the graph structures and embeddings in order to perform well in downstream tasks. Those potential applications range from computer vision, natural language processing, and network analysis. For instance, our research might be used to help better capture the semantic relationships between word tokens (beyond a sequence of tokens) in natural language processing.

There are many benefits of using our method as a tool, such as applying graph neural networks to non-graph structured data without manual graph construction, and learning node/graph embeddings that are more robust to noisy input graphs. Those benefits that might be utilized by a large number of potential applications may have a board range of societal impacts:

- the use of our research could improve and speed up the process of learning meaningful graphs from noisy/incomplete graphs (e.g., social networks) or even non-graph structured data (e.g., text and images).
- the use of our research could improve the robustness of graph neural networks to noisy/incomplete graph-structured data in terms of learning good node/graph embeddings for downstream task.

We would encourage the research to explore similar approaches in more specific real-world applications. We would also suggest the research to understand the adversarial robustness of the use of graph neural networks in safety/security-critical applications.

Acknowledgments and Disclosure of Funding

This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) through Cooperative Agreement D20AC00004 awarded by the U.S. Department of the Interior (DOI), Interior Business Center. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. This work is also supported by IBM Research AI through the IBM AI Horizons Network. We thank the anonymous reviewers for their constructive suggestions.

References

[1] X. Bai, L. Zhu, C. Liang, J. Li, X. Nie, and X. Chang. Multi-view feature selection via nonnegative structured graph learning. *Neurocomputing*, 2020.

[2] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.

[3] P. Berger, G. Hannak, and G. Matz. Efficient graph learning from noisy and incomplete data. *IEEE Transactions on Signal and Information Processing over Networks*, 6:105–119, 2020.

[4] A. Bojchevski, Y. Matkovic, and S. Günnemann. Robust spectral clustering for noisy data: Modeling sparse corruptions improves latent embeddings. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 737–746, 2017.

[5] H. Chen, L. Wang, S. Wang, D. Luo, W. Huang, and Z. Li. Label aware graph convolutional network–not all edges deserve your attention. *arXiv preprint arXiv:1907.04707*, 2019.

[6] Y. Chen, L. Wu, and M. J. Zaki. Graphflow: Exploiting conversation flow with graph neural networks for conversational machine comprehension. *arXiv preprint arXiv:1908.00059*, 2019.

[7] Y. Chen, L. Wu, and M. J. Zaki. Reinforcement learning based graph-to-sequence model for natural question generation. *arXiv preprint arXiv:1908.04942*, 2019.

[8] E. Choi, Z. Xu, Y. Li, M. W. Dusenberry, G. Flores, Y. Xue, and A. M. Dai. Graph convolutional transformer: Learning the graphical structure of electronic health records. *arXiv preprint arXiv:1906.04716*, 2019.

[9] J. Cussens. Bayesian network learning with cutting planes. *arXiv preprint arXiv:1202.3713*, 2012.

[10] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst. Learning laplacian matrix in smooth graph signal representations. *IEEE Transactions on Signal Processing*, 64(23):6160–6173, 2016.

[11] D. Dua and C. Graff. UCI machine learning repository, 2017.

[12] H. E. Egilmez, E. Pavez, and A. Ortega. Graph learning from data under laplacian and structural constraints. *IEEE Journal of Selected Topics in Signal Processing*, 11(6):825–841, 2017.

[13] P. Elinas, E. V. Bonilla, and L. Tiao. Variational inference for graph convolutional networks in the absence of graph data and adversarial settings. *arXiv*, pages arXiv–1906, 2019.

[14] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis. All you need is low (rank) defending against adversarial attacks on graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 169–177, 2020.

[15] L. Franceschi, M. Niepert, M. Pontil, and X. He. Learning discrete structures for graph neural networks. *arXiv preprint arXiv:1903.11960*, 2019.

[16] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[17] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

[18] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] F. Hu, Y. Zhu, S. Wu, L. Wang, and T. Tan. Semi-supervised node classification via hierarchical graph convolutional networks. *arXiv preprint arXiv:1902.06667*, 2019.

[21] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[22] S. Huang, Z. Kang, I. W. Tsang, and Z. Xu. Auto-weighted multi-view clustering via kernelized graph learning. *Pattern Recognition*, 88:174–184, 2019.

[23] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11313–11320, 2019.

[24] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang. Graph structure learning for robust graph neural networks. *arXiv preprint arXiv:2005.10203*, 2020.

[25] V. Kalofolias. How to learn a graph from smooth signals. In *Artificial Intelligence and Statistics*, pages 920–929, 2016.

[26] V. Kalofolias and N. Perraudin. Large scale graph learning from smooth signals. *arXiv preprint arXiv:1710.05654*, 2017.

[27] Z. Kang, H. Pan, S. C. Hoi, and Z. Xu. Robust graph learning from noisy data. *IEEE transactions on cybernetics*, 2019.

[28] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[30] J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[31] J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems*, pages 13333–13345, 2019.

[32] K. Lang. Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*, pages 331–339. Elsevier, 1995.

[33] Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[34] R. Li, S. Wang, F. Zhu, and J. Huang. Adaptive graph convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[35] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[36] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *International Conference on Learning Representations*, 2016.

[37] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[38] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pages 4257–4267, 2019.

[39] P. Liu, S. Chang, X. Huang, J. Tang, and J. C. K. Cheung. Contextualized non-local neural networks for sequence learning. *arXiv preprint arXiv:1811.08600*, 2018.

[40] S. Liu, Y. Chen, X. Xie, J. K. Siow, and Y. Liu. Automatic code summarization via multi-dimensional semantic fusing in gnn. *arXiv preprint arXiv:2006.05405*, 2020.

[41] W. Liu, J. He, and S.-F. Chang. Large graph construction for scalable semi-supervised learning. In *ICML*, 2010.

[42] L. Lovász. Random walks on graphs: A survey. Department of Computer Science, Yale University, 1994.

[43] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang. Graph convolutional networks with eigenpooling. *arXiv preprint arXiv:1904.13107*, 2019.

[44] H. V. Nguyen and L. Bai. Cosine similarity metric learning for face verification. In *Asian conference on computer vision*, pages 709–720. Springer, 2010.

[45] W. Norcliffe-Brown, S. Vafeias, and S. Parisot. Learning conditioned graph structures for interpretable visual question answering. In *Advances in Neural Information Processing Systems*, pages 8344–8353, 2018.

[46] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

[47] B. Samanta, A. De, N. Ganguly, and M. Gomez-Rodriguez. Designing random graph models using variational autoencoders with applications to chemical design. *arXiv preprint arXiv:1802.05283*, 2018.

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

[49] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. *arXiv preprint arXiv:2001.09382*, 2020.

[50] L. Sun, Y. Dou, C. Yang, J. Wang, P. S. Yu, and B. Li. Adversarial attack and defense on graph data: A survey. *arXiv preprint arXiv:1812.10528*, 2018.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[52] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[53] Y. Wang, Z. Zhao, and Z. Feng. Graspel: Graph spectral learning at scale. *arXiv preprint arXiv:1911.10373*, 2019.

[54] N. Wojke and A. Bewley. Deep cosine metric learning for person re-identification. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 748–756. IEEE, 2018.

[55] L. Wu, I. E.-H. Yen, Z. Zhang, K. Xu, L. Zhao, X. Peng, Y. Xia, and C. Aggarwal. Scalable global alignment graph kernel using random features: From node embedding to graph embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1418–1428, 2019.

[56] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

[57] K. Xu, L. Wu, Z. Wang, and V. Sheinin. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*, 2018.

[58] K. Xu, L. Wu, Z. Wang, M. Yu, L. Chen, and V. Sheinin. Exploiting rich syntactic information for semantic parsing with graph-to-sequence model. *arXiv preprint arXiv:1808.07624*, 2018.

[59] D.-Y. Yeung and H. Chang. A kernel approach for semisupervised metric learning. *IEEE Transactions on Neural Networks*, 18(1):141–149, 2007.

[60] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[61] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

[62] Y. Yu, J. Chen, T. Gao, and M. Yu. Dag-gnn: Dag structure learning with graph neural networks. *arXiv preprint arXiv:1904.10098*, 2019.

[63] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim. Graph transformer networks. In *Advances in Neural Information Processing Systems*, pages 11960–11970, 2019.

[64] Y. Zhang, S. Pal, M. Coates, and D. Ustebay. Bayesian graph convolutional neural networks for semi-supervised classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5829–5836, 2019.

[65] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang. Robust graph representation learning via neural sparsification. In *ICML*, 2020.

[66] X. Zheng, B. Aragam, P. K. Ravikumar, and E. P. Xing. Dags with no tears: Continuous optimization for structure learning. In *Advances in Neural Information Processing Systems*, pages 9472–9483, 2018.

A Theoretical Model Analysis

A.1 Theoretical Proof of Recovering Node and Anchor Graphs from Affinity Matrix $\mathbf{R}$

It is worth noting that a node-anchor affinity matrix $\mathbf{R}$ serves as a weighted adjacency matrix of a bipartite graph $\mathcal{B}$. We hence establish stationary Markov random walks [42] by defining the one-step transition probabilities as follows,

$$p^{(1)}(u_k|v_i) = \frac{R_{ik}}{\sum_{k'=1}^{s} R_{ik'}}, \quad p^{(1)}(v_i|u_k) \quad = \frac{R_{ik}}{\sum_{i'=1}^{n} R_{i'k}}, \quad \forall v_i \in \mathcal{V}, \quad \forall u_k \in \mathcal{U} \tag{12}$$

We can further compute the two-step transition probabilities between nodes as follows,

$$p^{(2)}(v_j|v_i) = \sum_{k=1}^{s} p^{(1)}(v_j|u_k)p^{(1)}(u_k|v_i) = \sum_{k=1}^{s} \frac{R_{jk}}{\sum_{j'=1}^{n} R_{j'k}} \frac{R_{ik}}{\sum_{k'=1}^{s} R_{ik'}} = \sum_{k=1}^{s} \frac{R_{jk}}{\Lambda_{kk}} \frac{R_{ik}}{\Delta_{ii}} \tag{13}$$

where $\Lambda_{kk} = \sum_{j'=1}^{n} R_{j'k}$ and $\Delta_{ii} = \sum_{k'=1}^{s} R_{ik'}$. Therefore, we can recover a row-normalized adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ for the node graph as $A_{ij} = p^{(2)}(v_j|v_i)$, which can be further written in a compact form $\mathbf{A} = \mathbf{\Delta}^{-1}\mathbf{R}\mathbf{\Lambda}^{-1}\mathbf{R}^\top$.

Similarly, we can compute the two-step transition probabilities between anchors as follows,

$$p^{(2)}(u_r|u_k) = \sum_{i=1}^{n} p^{(1)}(u_r|v_i)p^{(1)}(v_i|u_k) = \sum_{i=1}^{n} \frac{R_{ir}}{\sum_{r'=1}^{s} R_{ir'}} \frac{R_{ik}}{\sum_{i'=1}^{n} R_{i'k}} = \sum_{i=1}^{n} \frac{R_{ir}}{\Delta_{ii}} \frac{R_{ik}}{\Lambda_{kk}} \tag{14}$$

And a row-normalized adjacency matrix $\mathbf{B} \in \mathbb{R}^{s \times s}$ for the anchor graph $\mathcal{Q}$ can be formulated as $B_{kr} = p^{(2)}(u_r|u_k)$. And we can obtain $\mathbf{B} = \mathbf{\Lambda}^{-1}\mathbf{R}^\top\mathbf{\Delta}^{-1}\mathbf{R}$.

A.2 Theoretical Convergence Analysis

While it is challenging to theoretically prove the convergence of the proposed iterative learning procedure due to the arbitrary complexity of the model, here we want to conceptually understand why it works in practice. Fig. 5 shows the information flow of the learned adjacency matrix $\mathbf{A}$ and the updated node embedding matrix $\mathbf{Z}$ during the iterative procedure. For the sake of simplicity, we omit some other variables such as $\widetilde{\mathbf{A}}$. As we can see, at t-th iteration, $\mathbf{A}^{(t)}$ is computed based on $\mathbf{Z}^{(t-1)}$ (Line 9), and $\mathbf{Z}^{(t)}$ is computed based on $\widetilde{\mathbf{A}}^{(t)}$ (Line 11) which is computed based on $\mathbf{A}^{(t)}$ (Eq. (3)). We further denote the difference between the adjacency matrices at the t-th iteration and the previous iteration by $\delta_A^{(t)}$. Similarly, we denote the difference between the node embedding matrices at the t-th iteration and the previous iteration by $\delta_Z^{(t)}$.

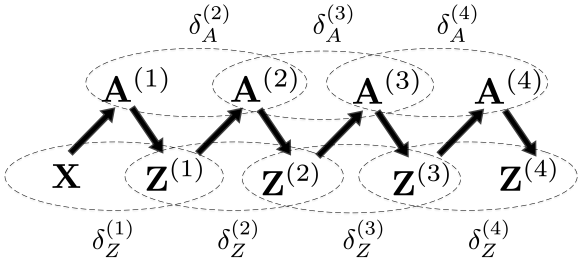


Figure 5: Information flow of iterative learning procedure.

If we assume that $\delta_Z^{(2)} < \delta_Z^{(1)}$, then we can expect that $\delta_A^{(3)} < \delta_A^{(2)}$ because conceptually a more similar node embedding matrix (i.e., smaller δ_Z) is supposed to produce a more similar adjacency matrix (i.e., smaller δ_A) given the fact that model parameters keep the same through iterations. Similarly, given that $\delta_A^{(3)} < \delta_A^{(2)}$, we can expect that $\delta_Z^{(3)} < \delta_Z^{(2)}$. Following this chain of reasoning, we can easily extend it to later iterations. In order to see why the assumption $\delta_Z^{(2)} < \delta_Z^{(1)}$ makes sense in practice, we need to recall the fact that $\delta_Z^{(1)}$ measures the difference between $\mathbf{Z}^{(1)}$ and $\mathbf{X}$, which is

usually larger than the difference between $\mathbf{Z}^{(2)}$ and $\mathbf{Z}^{(1)}$, namely $\delta_Z^{(2)}$. For example, the raw node feature matrix $\mathbf{X}$ can be quite sparse in practice (e.g., in Cora and Citeseer), whereas $\mathbf{Z}^{(1)}$ is typically a dense matrix.

A.3 Model Complexity Analysis

As for IDGL, the cost of learning an adjacency matrix is $\mathcal{O}(n^2h)$ for n nodes and data in $\mathbb{R}^h$, while computing node embeddings costs $\mathcal{O}(n^2h + ndh)$, computing task output costs $\mathcal{O}(n^2d)$, and computing the total loss costs $\mathcal{O}(n^2d)$ where d is the hidden size. We set the maximal number of iterations to T, hence the overall complexity is $\mathcal{O}(Tn(nh + nd + hd))$. If we assume that $d \approx h$ and $n \gg d$, the overall time complexity is $\mathcal{O}(Tdn^2)$.

As for IDGL-ANCH, the cost of learning a node-anchor affinity matrix is $\mathcal{O}(nsh)$, while computing node embeddings costs $\mathcal{O}(nsh + ndh + |\mathcal{E}|h)$, computing task output costs $\mathcal{O}(nsd + |\mathcal{E}|d)$, and computing the total loss costs $\mathcal{O}(ns^2 + s^2d)$ where $|\mathcal{E}|$ is the number of edges in the initial or kNN graph $\mathcal{G}$. With the assumption that the initial or kNN graph is usually very sparse in practice, especially for large graphs, we hence set $|\mathcal{E}| = kn$ where k is a constant denoting the average degree of the initial or kNN graph. Therefore, we get the overall time complexity $\mathcal{O}(Tn(ds + d^2 + s^2))$. If we assume that $n \gg s$ which usually holds true for large graphs, the overall time complexity is linear with respect to the numbers of graph nodes n.

As for space complexity, compared to IDGL, IDGL-ANCH reduces it from $\mathcal{O}(n^2)$ to $\mathcal{O}(ns)$ since it only needs to store the $n \times s$ affinity matrix.

B Empirical Model Analysis

B.1 Convergence Test

Here, we show the evolution of the learned adjacency matrix and accuracy through iterations in the iterative learning procedure in the testing phase. As we can see, both the adjacency matrix and accuracy converge quickly.

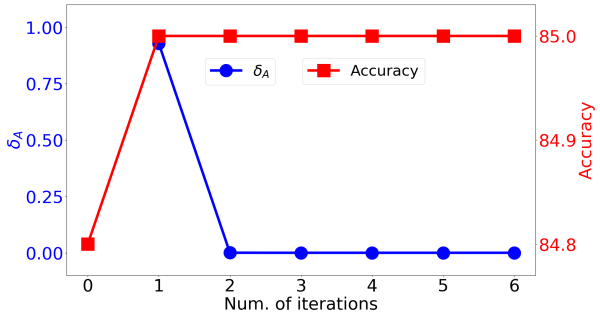


Figure 6: Convergence study on Cora (single run results).

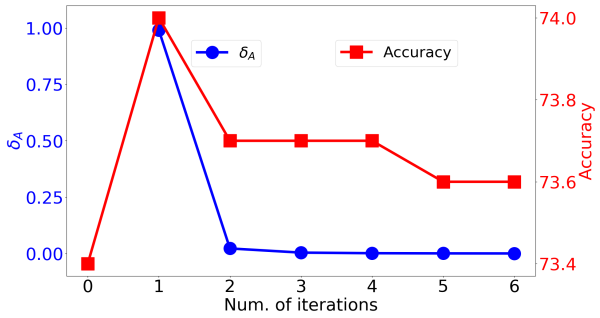


Figure 7: Convergence study on Citeseer (single run results).

B.2 Stopping Strategy Analysis

Here, we empirically compare the effectiveness of two stopping strategies: i) using a fixed number of iterations (blue line), and ii) using a stopping criterion to dynamically determine the convergence (red line). As we can see, dynamically adjusting the number of iterations using the stopping criterion works better in practice. Compared to using a fixed number of iterations globally, the advantage of applying this dynamical stopping strategy becomes more clear when we are doing mini-batch training since we can adjust when to stop dynamically for each example graph in the mini-batch.

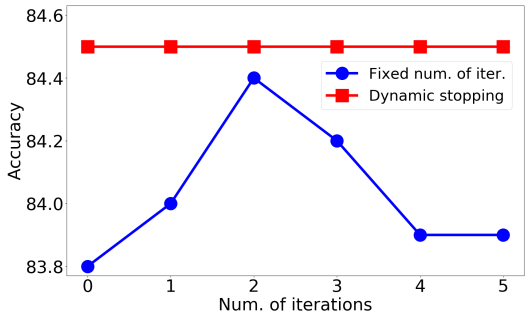


Figure 8: Performance comparison (i.e., test accuracy in %) of two different stopping strategies on Cora.

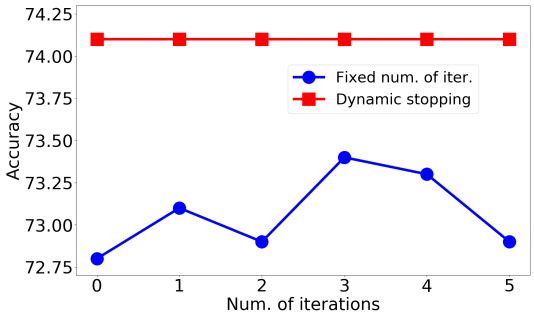


Figure 9: Performance comparison (i.e., test accuracy in %) of two different stopping strategies on Citeseer.

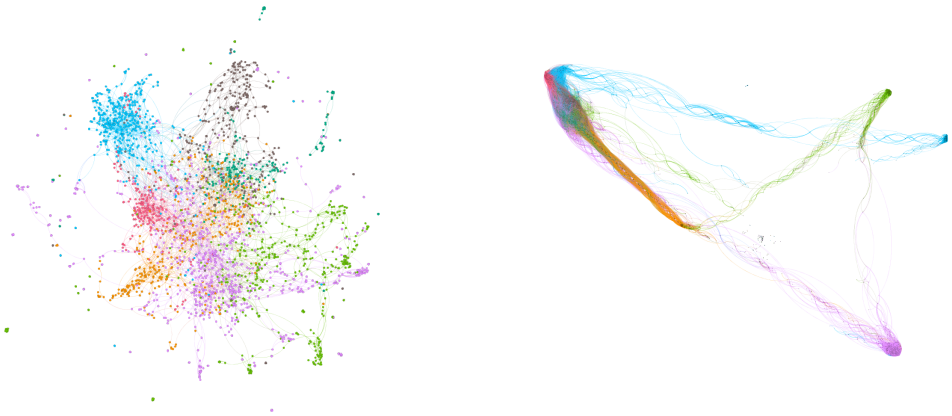
B.3 Graph Visualization

Here, we visualize the graph structures (i.e., $\mathbf{A}^{(t)}$) learned by IDGL. As we can see, compared to the initial graph structures, IDGL mainly forms graph structures within the same class of nodes, which complement the initial graph structure. This is as expected because $\mathbf{A}^{(t)}$ is computed based on the updated node embeddings that are supposed to capture certain node label information.

B.4 Hyperparameter Analysis

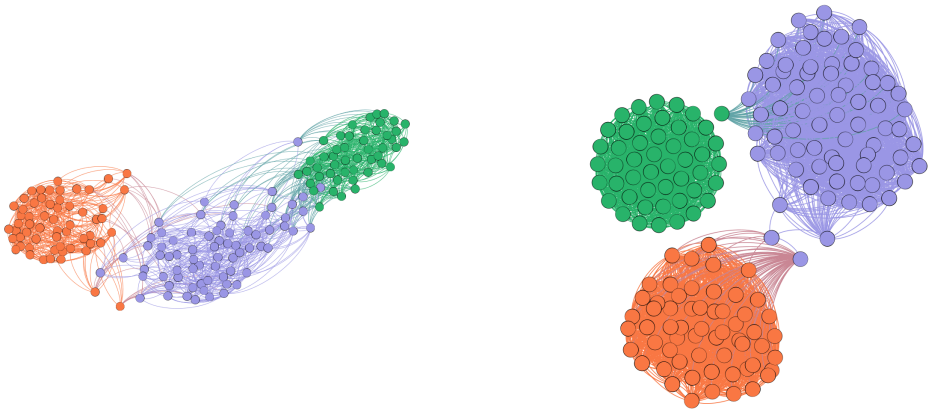
A hyperparameter λ is used to balance the trade-off between using the learned graph structure and the initial (or kNN) graph structure. In Table 5, we show the results of using different values of λ on Cora.

We also study the effect of the hyperparameter s (i.e., the number of anchors in IDGL-ANCH). As shown in Table 6, lower value of s can degrade the performance of IDGL-ANCH whereas after certain optimal value, further increasing the number of anchors might not help the performance.



(a) Initial graph ($\mathbf{A}^{(0)}$) (b) Learned graph ($\mathbf{A}^{(t)}$)

Figure 10: Visualization of the initial graph and the learned graph on Cora. Colors indicate different node labels.



(a) kNN graph ($\mathbf{A}^{(0)}$) (b) Learned graph ($\mathbf{A}^{(t)}$)

Figure 11: Visualization of the kNN graph and the learned graph on Wine. Colors indicate different node labels.

Table 5: Test scores ($\pm$ standard deviation) with different values of λ on the Cora data.

Methods / λ	0.9	0.8	0.7	0.6	0.5
IDGL	83.6 (0.4)	**84.5 (0.3)**	83.9 (0.3)	82.4 (0.1)	80.9 (0.2)
IDGL-ANCH	83.2 (0.4)	**84.4 (0.2)**	83.5 (0.6)	82.9 (0.4)	54.6 (32.3)

Table 6: Test scores ($\pm$ standard deviation) with different values of s for IDGL-ANCH on the Cora and Pubmed data.

Methods / s	1,600	1,300	1,000	700	400	100
Cora	84.0 (0.4)	84.1 (0.5)	**84.4 (0.2)**	83.8 (0.2)	58.7 (30.5)	38.3 (25.9)
Pubmed	82.7 (0.2)	**83.0 (0.4)**	82.7 (0.4)	**83.0 (0.2)**	82.7 (0.3)	82.4 (0.5)

C Details on Experimental Setup

C.1 Data Statistics

Table 7 shows the data statistics of the nine benchmarks used in our experiments.

Table 7: Data statistics. (clf. indicates classification and reg. indicates regression.)

Benchmarks	#Nodes	#Edges	Train/Dev/Test	Task	Setting
Cora	2,708 (1 graph)	5,429	140/500/1,000	node clf.	transductive
Citeseer	3,327 (1 graph)	4,732	120/500/1,000	node clf.	transductive
Pubmed	19,717 (1 graph)	44,338	60/500/1,000	node clf.	transductive
ogbn-arxiv	169,343 (1 graph)	1,166,243	90,941/29,799/48,603	node clf.	transductive
Wine	178 (1 graph)	N/A	10/20/158	node clf.	transductive
Cancer	569 (1 graph)	N/A	10/20/539	node clf.	transductive
Digits	1,797 (1 graph)	N/A	50/100/1,647	node clf.	transductive
20News	317 (18,846 graphs)	N/A	7,919/3,395/7,532	graph clf.	inductive
MRD	389 (5,006 graphs)	N/A	3,003/1,001/1,002	graph reg.	inductive

C.2 Model Settings

Table 8: Hyperparameter for IDGL on all benchmarks.

Benchmarks	λ	η	α	β	γ	k	ϵ	m	δ	T
Cora	0.8	0.1	0.2	0.0	0.0	–	0.0	4	4.0e-5	10
Citeseer	0.6	0.5	0.4	0.0	0.2	–	0.3	1.0	1.0e-3	10
Wine	0.8	0.7	0.1	0.1	0.3	20	0.75	1	1.0e-3	10
Cancer	0.25	0.1	0.4	0.2	0.1	40	0.9	1	1.0e-3	10
Digits	0.4	0.1	0.4	0.1	0.0	24	0.65	8	1.0e-4	10
20News	0.1	0.4	0.5	0.01	0.3	950	0.3	12	8.0e-3	10
MRD	0.5	0.9	0.2	0.0	0.1	350	0.4	5	4.0e-2	10

Table 9: Hyperparameter for IDGL-ANCH on all benchmarks.

Benchmarks	λ	η	α	β	γ	k	ϵ	m	δ	T	num./ratio of anchors
Cora	0.8	0.1	0.2	0.0	0.1	–	0.0	4	8.5e-5	10	1,000
Citeseer	0.6	0.5	0.5	0.1	0.2	–	0.2	4	2.0e-3	10	1,400
Pubmed	0.7	0.3	0.0	0.03	0.0	–	0.1	6	8.0e-5	10	700
ogbn-arxiv	0.8	0.1	0.2	0.0	0.0	–	0.9	1	1.0e-1	2	300
Wine	0.7	0.7	0.1	0.1	0.3	20	0.75	1	1.0e-3	10	200
Cancer	0.25	0.1	0.0	0.0	0.0	40	0.9	4	8.0e-4	10	100
Digits	0.3	0.3	0.4	0.1	0.0	24	0.65	8	1.0e-4	10	1,500
20News	0.1	0.3	0.4	0.0	0.3	950	0.4	12	1.0e-2	10	0.4
MRD	0.5	0.75	0.2	0.0	0.0	400	0.7	4	3.0e-2	10	0.4

In all our experiments, we apply a dropout ratio of 0.5 after GCN layers except for the output GCN layer. During the iterative learning procedure, we also apply a dropout ratio of 0.5 after the intermediate GCN layer, except for Citeseer (no dropout) and Digits (0.3 dropout). For experiments on text benchmarks, we keep and fix the 300-dim GloVe vectors for words that appear more than 10 times in the dataset. For long documents, for the sake of efficiency, we cut the text length to maximum 1,000 words. We apply a dropout ratio of 0.5 after word embedding layers and BiLSTM layers. The batch size is set to 16. And the hidden size is set to 128 and 64 for 20News and MRD, respectively. For all other benchmarks, the hidden size is set to 16 to follow the original GCN paper. For the text benchmarks, we apply a BiLSTM to a sequence of word embeddings. The concatenation of the last forward and backward hidden states of the BiLSTM is used as the initial node features. We use Adam [28] as the optimizer. For the text benchmarks, we set the learning rate to 1e-3. For all other benchmarks, we set the learning rate to 0.01 and apply L2 norm regularization with weight decay set to 5e-4. As for IDGL-ANCH, we set the number of anchors as a hyperparameter in transductive experiments, while in inductive experiments, we set the ratio of anchors (proportional to the graph

size) as a hyperparameter. In Table 8 and Table 9, we show the hyperparameters for IDGL and IDGL-ANCH on all benchmarks, respectively. All hyperparameters are tuned on the development set.