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 Iterative Deep Graph Learning (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 [27, 54, 16, 49], graph classification [57, 41] and graph generation [45, 35, 58]. 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, 54, 55] 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 Iterative Deep Graph Learning (IDGL), for jointly and iteratively learning the graph structure and the GNN parameters that are optimized towards the downstream prediction task. The key rationale of our
IDGL 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 supervised 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 prediction 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 [23] 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 $G = (\mathcal{V}, \mathcal{E})$ be represented as a set of $n$ nodes $v_i \in \mathcal{V}$ with an initial node feature matrix $X \in \mathbb{R}^{d \times n}$, edges $(v_i, v_j) \in \mathcal{E}$ (binary or weighted) formulating an initial noisy adjacency matrix $A^{(0)} \in \mathbb{R}^{n \times n}$, and a degree matrix $D^{(0)} = \sum_j A^{(0)}_{ij}$. Given a noisy graph input $G := \{A^{(0)}, X\}$ or only a feature matrix $X \in \mathbb{R}^{d \times n}$, the deep graph learning problem we consider in this paper is to produce an optimized graph $G^* := \{A^*, X\}$ and its corresponding graph node embeddings $Z = f(G^*, \theta) \in \mathbb{R}^{h \times n}$, 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 $X$ 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 [49] 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.

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 supervised prediction tasks. The iterative method adjusts when to stop 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 constructing such a graph can be optimized towards the learning task at hand and the whole learning system is end-to-end trainable.

2.3 Graph Learning as Similarity Metric Learning

Previous methods (e.g., [14]) 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 [42, 51], radial basis function (RBF) kernel [56, 32] and attention mechanisms [48, 22]. 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,

\[ s_{ij}^p = \cos(w_p \odot v_i, w_p \odot v_j), \quad s_{ij} = \frac{1}{m} \sum_{p=1}^{m} s_{ij}^p \]  

Intuitively, \( s_{ij}^p \) computes the cosine similarity between the two input vectors \( v_i \) and \( v_j \), for the \( p \)-th perspective, where each perspective considers one part of the semantics captured in the vectors.

Graph sparsification via \( \varepsilon \)-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 $A$ from $S$ by considering only the $\varepsilon$-neighborhood for each node. Specifically, we mask off (i.e., set to zero) those elements in $S$ which are smaller than a non-negative threshold $\varepsilon$.

**Anchor-based scalable metric learning.** The above similarity metric function like Eq. (1) computes similarity scores for all pairs of graph nodes, which requires $O(n^2)$ complexity for both computational time and memory consumption, rendering significant scalability issue for large graphs. To address the scalability issue, inspired by previous anchor-based methods [39, 52], we design an anchor-based scalable metric learning technique which learns a node-anchor affinity matrix $R \in \mathbb{R}^{n \times n}$ (i.e., requires $O(ns)$ for both time and space complexity where $s$ is the number of anchors) between the node set $V$ and the anchor set $U$. Note that $s$ is a hyperparameter which is tuned on the development set.

Specifically, we randomly sample a set of $s \in U$ anchors from the node set $V$, 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:

$$a_{ik}^p = \cos(w_p \odot v_i, w_p \odot u_k), \quad a_{ik} = \frac{1}{m} \sum_{p=1}^{m} a_{ik}^p \quad (2)$$

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

### 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 $A$ could be supplementary to the original graph topology $A^{(0)}$ 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,

$$\hat{A}^{(t)} = \lambda L^{(0)} + (1 - \lambda) \left\{ \eta f(A^{(t)}) + (1 - \eta) f(A^{(1)}) \right\} \quad (3)$$

where $L^{(0)} = D^{(0)} A^{(0)} D^{(0)}$ is the normalized adjacency matrix of the initial graph. $A^{(t)}$ and $A^{(1)}$ 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, $f(A)_{ij} = A_{ij} / \sum_j A_{ij}$.

Note that $A^{(1)}$ is computed from the raw node features $X$, whereas $A^{(t)}$ is computed from the previously updated node embeddings $Z^{(t-1)}$ that is optimized towards the downstream prediction task. Therefore, we make the final learned graph structure as their linear combination weighted by a hyperparameter $\eta$, so as to combine the advantages of both. Finally, another hyperparameter $\lambda$ 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 $X$ using cosine similarity.

Our graph learning framework is agnostic to various GNN architectures and prediction tasks. In this paper, we adopt a two-layered GCN [27] where the first layer (denoted as GNN$_1$) maps the raw node features $X$ to the intermediate embedding space, and the second layer (denoted as GNN$_2$) further maps the intermediate node embeddings $Z$ to the output space.

$$Z = \text{ReLU}(\text{MP}(X, \hat{A}) W_1), \quad \hat{y} = \sigma(\text{MP}(Z, \hat{A}) W_2), \quad \mathcal{L}_{\text{pred}} = \ell(\hat{y}, y) \quad (4)$$

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. $\text{MP}(\cdot, \cdot)$ is a message passing function, and in GCN, $\text{MP}(\cdot, \cdot) = \hat{A} F$ for a feature/embedding matrix $F$ and normalized adjacency matrix $\hat{A}$ which we obtain using Eq. (3).

**Node-anchor message passing.** Note that a node-anchor affinity matrix $R$ serves as a weighted adjacency matrix of a bipartite graph $B$ 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 $R$, built
upon theories of stationary Markov random walks [40], we can actually recover both the node graph \( G \) and the anchor graph \( Q \) from \( R \) by computing the two-step transition probabilities. Let \( A \in \mathbb{R}^{n \times n} \) denote a row-normalized adjacency matrix for the node graph \( G \), and \( A_{ij} = p^{(2)}(v_j | v_i) \) indicate the two-step transition probability from node \( v_i \) to node \( v_j \). \( A \) can be recovered from \( R \),

\[
A = \Delta^{-1} R \Lambda^{-1} R^T
\]  

(5)

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

\[
B = \Lambda^{-1} R^T \Delta^{-1} R
\]  

(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 \( A \) from \( R \) (Eq. (5)) and directly performing message passing over the node graph \( G \) (Eq. (4)) are expensive in both time complexity (\( O(n^2 s) \)) and space complexity (\( O(n^2 s) \)), 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 \( \tilde{B} \), formulated as follows,

\[
MP_{12}(F, R) = MP_2(F', R), \quad F' = MP_1(F, R)
\]  

(7)

where \( MP_1(F, R) = \Delta^{-1} R^T F \) aims to pass message \( F \) from the nodes \( \mathcal{V} \) to the anchors \( \mathcal{U} \), and \( MP_2(F', R) = \Delta^{-1} R F' \) aims to further pass the message \( F' \) aggregated on the anchors back to the nodes. Finally, we can obtain \( MP_{12}(F, R) = \Delta^{-1} R A^{-1} R^T F = AF \) where \( A \) is the node adjacency matrix recovered from \( R \) using Eq. (5). In this way, we reduce both time and space complexity to \( O(ns) \). Therefore, we can rewrite the regular node embedding and prediction equations defined in Eqs. (3) and (4) as follows,

\[
Z = \text{ReLU}(\text{MP}_a(X, \{L^{(0)}, R^{(t)}, R^{(1)}\}) W_1), \quad \hat{y} = \sigma(\text{MP}_a(Z, \{L^{(0)}, R^{(t)}, R^{(1)}\}) W_2)
\]  

(8)

where \( \text{MP}_a(\cdot, \cdot) \) is a hybrid message passing function with the same spirit of Eq. (4), defined as,

\[
\text{MP}_a(F, \{L^{(0)}, R^{(t)}, R^{(1)}\}) = \lambda \text{MP}(F, L^{(0)}) + (1 - \lambda) \left\{ \eta \text{MP}_{12}(F, R^{(t)}) + (1 - \eta) \text{MP}_{12}(F, R^{(1)}) \right\}
\]  

(9)

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

2.5 Graph Regularization

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

Let each column of the feature matrix \( X \) 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 \( x_1, \ldots, x_n \in \mathbb{R}^d \) is usually measured by the Dirichlet energy [2],

\[
\Omega(A, X) = \frac{1}{2n^2} \sum_{i,j} A_{ij} ||x_i - x_j||^2 = \frac{1}{n^2} \text{tr}(X^T LX)
\]  

(10)

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

However, solely minimizing the smoothness loss will result in the trivial solution \( A = 0 \). Also, it is desirable to have control of how sparse the resulting graph is. Following [23], we impose additional constraints on the learned graph,

\[
f(A) = \frac{-\beta}{n} I^T \log(A 1) + \frac{\gamma}{n^2} ||A||_F^2
\]  

(11)
We then define the overall graph regularization loss as the sum of the above losses. As shown in Eq. (6), we can obtain a row-normalized adjacency matrix

Algorithm 1 General Framework for IDGL and IDGL-ANCH

1: Input: \( X, y, [A]^{(0)} \)
2: Parameters: \( m, \varepsilon, \alpha, \beta, \gamma, \lambda, \delta, T, \eta, k, s \)
3: Output: \( \Theta, \tilde{y}, \tilde{A}^{(t)} \) or \( R^{(t)} \)
4: \( [A]^{(0)} \leftarrow \text{kNN}(X, k) \) \( [\text{kNN-graph if no initial } A^{(0)}] \)
5: \( t \leftarrow 1 \)
6: StopCond \( \leftarrow ||A^{(t)} - A^{(t-1)}||_F^2 > \delta ||A^{(1)}||_F^2 \) if IDGL else \( ||R^{(t)} - R^{(t-1)}||_F^2 > \delta ||R^{(t)}||_F^2 \)
7: while \((t = 1 \text{ or StopCond}) \) and \( t \leq T \) do
8: if IDGL then
9: \( A^{(t)} \leftarrow \text{GL}(X) \) or \( \text{GL}(Z^{(t-1)}) \) using Eq. (1) \( \{\text{Refine adj. matrix}\} \)
10: \( \tilde{A}^{(t)} \leftarrow [A]^{(0)}, \tilde{A}^{(t)}, A^{(1)} \) using Eq. (4) \( \{\text{Combine refined and raw adj. matrices}\} \)
11: \( Z^{(t)} \leftarrow \text{GNN}_1(\tilde{A}^{(t)}, X) \) using Eq. (4) \( \{\text{Refine node embeddings}\} \)
12: else
13: \( R^{(t)} \leftarrow \text{GL}(X, X_{d}) \) or \( \text{GL}(Z^{(t-1)}, Z_t^{(t-1)}) \) using Eq. (5) \( \{\text{Refine affinity matrix}\} \)
14: \( Z^{(t)} \leftarrow \text{GNN}_1([A]^{(0)}, R^{(t)}, R^{(1)}), X) \) using Eqs. (8) and (9) \( \{\text{Refine node embeddings}\} \)
15: end if
16: \( \tilde{y} \leftarrow \text{GNN}_2(\tilde{A}^{(t)}, Z^{(t)}) \) using Eq. (4) if IDGL else \( \text{GNN}_2([A]^{(0)}, R^{(t)}, R^{(1)}), Z^{(t)}) \) using Eqs. (8) and (9)
17: \( L^{(t)}_{\text{pred}} \leftarrow \text{LOSS}_5(\tilde{y}, y) \) using Eq. (4)
18: \( L^{(t)}_\alpha \leftarrow \alpha \Omega[A]^{(t)}, X] + f(A^{(t)}) \) if IDGL else \( \alpha \Omega[\hat{B}^{(t)}, X^{(t)}] + f(\hat{B}^{(t)}) \) where \( \hat{B}^{(t)} = R^{(t)} \Delta^{-1} R^{(t)} \)
19: \( L^{(t)} = L^{(t)}_{\text{pred}} + L^{(t)}_\alpha \) and \( t \leftarrow t + 1 \)
20: end while
21: \( \hat{L} \leftarrow L^{(t)} + \sum_{t=2}^{T} L^{(t-1)}/(t - 1) \)
22: Back-propagate \( \hat{L} \) to update model weights \( \Theta \) \( \{\text{In training phase only}\} \)

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 \( L = \alpha \Omega[A, X] + f(A) \), which is able to control the smoothness, connectivity and sparsity of the learned graph where \( \alpha \), \( \beta \) and \( \gamma \) are all non-negative hyperparameters.

Anchor graph regularization. As shown in Eq. (6), we can obtain a row-normalized adjacency matrix \( B \) for the anchor graph \( Q \) in \( \mathcal{O}(ns^2) \) time complexity. In order to control the quality of the learned node-anchor affinity matrix \( R \) (which can result in implicit control of the quality of the node adjacency matrix \( A \)), we apply the aforementioned graph regularization techniques to the anchor graph. It is worth noting that our proposed graph regularization loss is only applicable to non-negative and symmetric adjacency matrices [23]. Therefore, instead of applying graph regularization to \( A \) which is often not symmetric, we opt to apply graph regularization to its unnormalized version \( \hat{B} = R A^{-1} R \) as \( L = \alpha \Omega[\hat{B}, X^{(t)}] + f(\hat{B}) \), where \( X^{(t)} \) denotes the set of anchor embeddings sampled from the set of node embeddings \( X \).

2.6 Joint Learning with A Hybrid Loss

Compared to previous works which directly optimize the adjacency matrix based on either graph regularization loss [24], or task-dependent prediction loss [14], 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 \( \hat{L} = L_{\text{pred}} + L_G \).

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) [26, 20] 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)) [30, 44] 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 [27, 14, 20]. 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 [14] 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 [27], GAT [49], GraphSAGE [17], APPNP [28], H-GCN [19] and GDC [29] as baselines. For non-graph and text benchmarks where the graph topology is unavailable, we conceive various GNN\(k\)NN baselines (i.e., GCN\(k\)NN, GAT\(k\)NN and GraphSAGE\(k\)NN) where a kNN graph on the data set is constructed during preprocessing before applying a GNN model. For text benchmarks, we include a BiLSTM [18] 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, we can see that IDGL outperforms all baselines in 4 out of 5 benchmarks, which demonstrates its effectiveness. Compared to IDGL, IDGL-ANCH is more scalable and can achieve comparable or even better results. Besides, we can see that our graph learning method can greatly help the node classification task even when the graph topology is given. When the graph topology is not available, compared to various GNN\(k\)NN baselines, IDGL consistently achieves much better results on all datasets, which shows the power of jointly learning graph structures and GNN parameters. The results of inductive experiments are shown in Table 2. Unlike LDS which cannot handle inductive setting, the good performance on 20News and MRD verifies 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.

<table>
<thead>
<tr>
<th>Model</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
<th>ogbn-arxiv</th>
<th>Wine</th>
<th>Cancer</th>
<th>Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>81.5</td>
<td>70.3</td>
<td>79.0</td>
<td>71.7 (0.3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAT</td>
<td>83.0 (0.7)</td>
<td>72.5 (0.7)</td>
<td>79.0 (0.3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphSAGE</td>
<td>77.4 (1.0)</td>
<td>67.0 (1.0)</td>
<td>76.6 (0.8)</td>
<td>71.5 (0.3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APPNP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H-GCN</td>
<td>84.5 (0.5)</td>
<td>72.8 (0.5)</td>
<td>79.8 (0.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCN+GDC</td>
<td>83.6 (0.2)</td>
<td>73.4 (0.3)</td>
<td>78.7 (0.4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDS</td>
<td>84.1 (0.4)</td>
<td>75.0 (0.4)</td>
<td></td>
<td></td>
<td>97.3 (0.4)</td>
<td>94.4 (1.9)</td>
<td>92.5 (0.7)</td>
</tr>
<tr>
<td>GCN(k)NN*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAT(k)NN*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GraphSAGE(k)NN*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDS*</td>
<td>83.9 (0.6)</td>
<td>74.8 (0.3)</td>
<td></td>
<td></td>
<td>96.9 (1.4)</td>
<td>93.4 (2.4)</td>
<td>90.8 (2.5)</td>
</tr>
<tr>
<td>IDGL</td>
<td>84.5 (0.3)</td>
<td>74.1 (0.2)</td>
<td></td>
<td></td>
<td>97.8 (0.6)</td>
<td>95.1 (1.0)</td>
<td>93.1 (0.5)</td>
</tr>
<tr>
<td>IDGL-ANCH</td>
<td>84.4 (0.2)</td>
<td>72.0 (1.0)</td>
<td>83.0 (0.2)</td>
<td>72.0 (0.3)</td>
<td>98.1 (1.1)</td>
<td>94.8 (1.4)</td>
<td>93.2 (0.9)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Methods</th>
<th>20News</th>
<th>MRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiLSTM</td>
<td>80.0 (0.4)</td>
<td>53.1 (1.4)</td>
</tr>
<tr>
<td>GCN$_{ANN}$</td>
<td>81.3 (0.6)</td>
<td>60.1 (1.5)</td>
</tr>
<tr>
<td>IDGL</td>
<td>83.6 (0.4)</td>
<td>63.7 (1.8)</td>
</tr>
<tr>
<td>IDGL-ANCH</td>
<td>82.9 (0.3)</td>
<td>62.9 (0.4)</td>
</tr>
</tbody>
</table>

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

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

**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 $\lambda$ (i.e., tuned on the development set), we enforce the model to rely less on the initial noisy graph.

![Figure 3: Test accuracy (± standard deviation) in percent for the edge attack scenarios on Cora.](image1)

![Figure 4: Convergence and stopping strategy study on Cora (Single run results).](image2)
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 \( \delta \|_{\mathcal{F}} = \| A^{(t)} - A^{(t-1)} \|_{\mathcal{F}} \). 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 [53, 31], 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, 50, 25, 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, 21] 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, 62, 59], and graph generation [36, 47], which have a different goal from ours.

In the field of GNNs [27, 15, 17, 33, 60], there is a line of research on developing robust GNNs that are insensitive to adversarial graphs by leveraging attention-based methods [5], Bayesian methods [13, 61] and graph diffusion-based methods [29]. Recently, researchers have explored methods to automatically construct a graph of objects [43, 8, 22, 14, 38] or words [47, 6, 7] when applying GNNs to non-graph structured data. However, these methods merely optimize the graphs towards the downstream tasks without the explicit control of the quality of the learned graphs. More recently, [14] 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 [48] that also 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 the GNN parameters that are optimized towards the prediction task at hand. The proposed method is able to iteratively search for an implicit graph structure that better helps the prediction task. Our extensive experiments demonstrate the effectiveness and efficiency of the proposed model. In the future, we are planning to explore more effective and scalable techniques for handling more challenging scenario where both graph topology and node feature matrix 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


