Deep Iterative and Adaptive Learning for Graph Neural Networks

Yu Chen[1], **Lingfei Wu**[2*], **Mohammed J. Zaki**[1]

[1]Rensselaer Polytechnic Institute
[2]IBM Research
cheny39@rpi.edu, lwu@email.wm.edu, zaki@cs.rpi.edu

Abstract

In this paper, we propose an end-to-end graph learning framework, namely **D**eep **I**terative and **A**daptive **L**earning for **G**raph **N**eural **N**etworks (DIAL-GNN), for jointly learning the graph structure and graph embeddings simultaneously. We first cast the graph structure learning problem as a similarity metric learning problem and leverage an adapted graph regularization for controlling smoothness, connectivity and sparsity of the generated graph. We further propose a novel iterative method for searching for a hidden graph structure that augments the initial graph structure. Our iterative method dynamically stops when the learned graph structure approaches close enough to the optimal graph. Our extensive experiments demonstrate that the proposed DIAL-GNN model can consistently outperform or match state-of-the-art baselines in terms of both downstream task performance and computational time. The proposed approach can cope with both transductive learning and inductive learning.

Introduction

Recent years have seen a growing amount of interest in graph neural networks (GNNs) (Kipf and Welling 2016; Li et al. 2016; Hamilton, Ying, and Leskovec 2017), with successful applications in broad areas such as computer vision (Norcliffe-Brown, Vafeias, and Parisot 2018), natural language processing (Xu et al. 2018a; 2018b; 2018c) and healthcare informatics (Gao et al. 2019). Unfortunately, GNNs can only be used when graph-structured data is available. Many real-world applications naturally admit graph-structured data like social networks. However, it is questionable if these intrinsic graph-structures are optimal for the downstream tasks. More importantly, many applications such as those in natural language processing may only have non-graph structured data or even just the original feature matrix, requiring additional graph construction from the original data matrix to formulate graph data.

In the field of graph signal processing, researchers have explored various ways of learning graphs from data, but without considering the downstream tasks (Dong et al. 2016; Kalofolias 2016; Kalofolias and Perraudin 2017; Egilmez, Pavez, and Ortega 2017). Independently, there has been an increasing amount of work studying the dynamic model of interacting systems utilizing implicit interaction models (Sukhbaatar, Fergus, and others 2016; Hoshen 2017; Van Steenkiste et al. 2018; Kipf et al. 2018). However, these methods cannot be directly applicable to jointly learning the graph structure and graph representations when the graph is noisy or even not available. Recently, researchers have explored methods to automatically construct a graph (Choi et al. 2019; Li et al. 2018; Liu et al. 2018; Chen, Wu, and Zaki 2019a; 2019b) when applying GNNs to non-graph structured data. However, these methods merely optimize the graphs towards the downstream tasks without utilizing the techniques which have proven to be useful in graph signal processing.

More recently, (Franceschi et al. 2019) presented a new approach for jointly learning the graph and the parameters of GNNs by approximately solving a bilevel program. However, this approach has severe scalability issue since it needs to learn N^2 number of (Bernoulli) random variables to model joint probability distribution on the edges of the graph consisting of N number of vertices. More importantly, it can only be used for transductive setting, which means this method cannot consider new nodes during the testing.

To address these limitations, in this paper, we propose a **D**eep **I**terative and **A**daptive **L**earning for **G**raph **N**eural **N**etworks (DIAL-GNN) framework for jointly learning the graph structure and the GNN parameters that are optimized towards some prediction task. In particular, we present a graph learning neural network that casts a graph learning problem as a data-driven similarity metric learning task for constructing a graph. We then adapt techniques for learning graphs from smooth signals (Kalofolias 2016) to serve as graph regularization. More importantly, we propose a novel iterative method to search for a hidden graph structure that augments the initial graph structure towards an optimal graph for the (semi-)supervised prediction tasks. The proposed approach can cope with both transductive learning and inductive learning. Our extensive experiments demonstrate that our model can consistently outperform or match state-of-the-art baselines on various datasets.

*Corresponding author.

Approach

With this paper we address the challenging problem of automatic graph structure learning for GNNs. We are given a set of n objects V associated with a feature matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$ encoding the feature descriptions of the objects. The goal is to automatically learn the graph structure $\mathcal{G}$, typically in the form of an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, underlying the set of objects, which will be consumed by a GNN-based model for a downstream prediction task. Unlike most existing methods that construct graphs based on hand-crafted rules or features during preprocessing, our proposed DIAL-GNN framework formulates the problem as an iterative learning problem that jointly learns the graph structure and the GNN parameters iteratively in an end-to-end manner. The overall model architecture is shown in Fig. 1.

Graph Learning as Similarity Metric learning

A common strategy of graph construction is to first compute the similarity between pairs of nodes based on some metric, and then consume the constructed graph in a downstream task. Unlike these methods, in this work, we design a learnable metric function for graph structure learning, which will be jointly trained with a task-dependent prediction model.

Similarity Metric Learning After preliminary experiments, we design a multi-head weighted cosine similarity,

$$\begin{aligned} s_{ij}^k &= \cos(\mathbf{w}_k \odot \mathbf{v}_i, \mathbf{w}_k \odot \mathbf{v}_j) \\ s_{ij} &= \frac{1}{m} \sum_{k=1}^{m} s_{ij}^k \end{aligned} \tag{1}$$

where $\odot$ denotes the Hadamard product. Specifically, we use m weight vectors (each has the same dimension as the input vectors and represents one perspective) to compute m cosine similarity matrices independently and take their average as the final similarity $\mathbf{S}$. Intuitively, s_{ij}^k computes the cosine similarity between the two input vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, for the k-th perspective where each perspective considers one part of the semantics captured in the vectors. This idea of multi-head similarity is similar to those in multi-head attention (Vaswani et al. 2017; Veličković et al. 2017).

Graph Sparsification via ε-neighborhood An adjacency matrix (same for a metric) is supposed to be non-negative while 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 makes little sense for most applications. We hence proceed to extract a symmetric sparse adjacency matrix $\mathbf{A}$ from $\mathbf{S}$ by considering only the ε-neighborhood for each node. Specifically, we mask off those elements in $\mathbf{S}$ which are smaller than certain non-negative threshold ε.

$$\mathbf{A}_{ij} = \begin{cases} s_{ij} & s_{ij} > \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Graph Regularization

In graph signal processing (Shuman et al. 2013), a widely adopted assumption for graph signals is that values change smoothly across adjacent nodes. Given an undirected graph with 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 (Belkin and Niyogi 2002),

$$\Omega(\mathbf{A}, \mathbf{X}) = \frac{1}{2n^2} \sum_{i,j} \mathbf{A}_{ij} ||\mathbf{x}_i - \mathbf{x}_j||^2 = \frac{1}{n^2} \text{tr}(\mathbf{X}^T \mathbf{L} \mathbf{X}) \tag{3}$$

where $\text{tr}(\cdot)$ denotes the trace of a matrix, $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the graph Laplacian, and $\mathbf{D} = \sum_j \mathbf{A}_{ij}$ is the degree matrix. As can be seen, minimizing $\Omega(\mathbf{A}, \mathbf{X})$ forces adjacent nodes to have similar features, thus enforces smoothness of the graph signals on the graph associated to $\mathbf{A}$. However, solely minimizing the above 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 (Kalofolias 2016), we impose additional constraints to the learned graph,

$$f(\mathbf{A}) = \frac{-\beta}{n} \mathbf{1}^T \log(\mathbf{A}\mathbf{1}) + \frac{\gamma}{n^2} ||\mathbf{A}||_F^2 \tag{4}$$

where $||\cdot||_F$ denotes the Frobenius norm. As we can see, 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.

In this work, we borrow the above techniques, and apply them as regularization to the graph learned by Eqs. (1) and (2). The overall graph regularization loss is defined as the sum of the above losses, which is able to control the smoothness, connectivity and sparsity of the resulting graph where α, β and γ are all non-negative hyperparameters.

$$\mathcal{L}_{\mathcal{G}} = \alpha \Omega(\mathbf{A}, \mathbf{X}) + f(\mathbf{A}) \tag{5}$$

An Iterative Graph Learning Method

Joint Graph Structure and Representation Learning We expect the graph structure underlying a set of objects to serve two purposes: i) it should respect the semantic relations among the objects, which is enforced by the metric function (Eq. (1)) and the smoothness loss (Eq. (3)); ii), it should suit the needs of the downstream prediction task. Compared to previous works which directly optimize the adjacency matrix based on either some graph regularization loss (Kalofolias and Perraudin 2017), or some task-dependent prediction loss (Franceschi et al. 2019), we propose to learn by minimizing a joint loss function combining both the task prediction loss and the graph regularization loss, namely, $\mathcal{L} = \mathcal{L}_{\text{pred}} + \mathcal{L}_{\mathcal{G}}$.

Note that our graph learning framework is agnostic to various GNNs and prediction tasks. In this paper, we adopt a two-layered GCN (Kipf and Welling 2016) where the first layer maps the node features to the node embedding space (Eq. (6)), and the second layer further maps the intermediate node embeddings to the output space (Eq. (7)).

$$\mathbf{Z} = \text{ReLU}(\widetilde{\mathbf{A}} \mathbf{X} \mathbf{W}_1) \tag{6}$$

$$\widehat{\mathbf{y}} = \sigma(\widetilde{\mathbf{A}} \mathbf{Z} \mathbf{W}_2) \tag{7}$$

$$\mathcal{L}_{\text{pred}} = \ell(\widehat{\mathbf{y}}, \mathbf{y}) \tag{8}$$

where $\widetilde{\mathbf{A}}$ is the normalized adjacency matrix, $\sigma(\cdot)$ is a task-dependent output function, and $\ell(\cdot)$ is a task-dependent loss

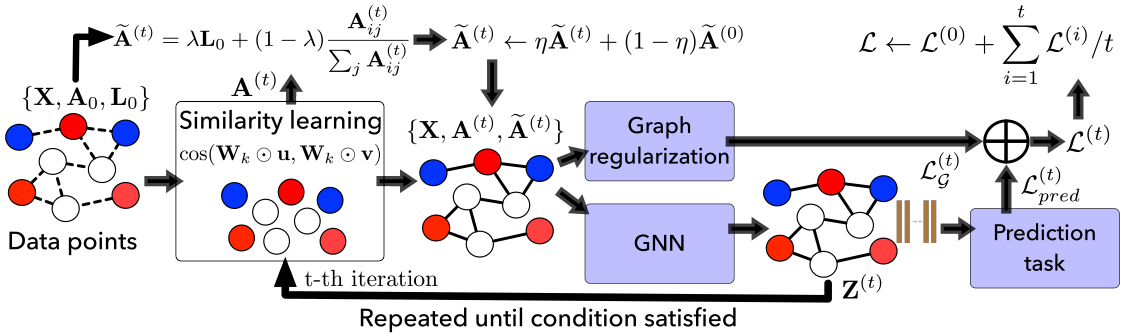

Figure 1: Overview of the proposed model. Dashed lines in the leftmost data points indicate the initial graph topology $\mathbf{A}_0$ either from the ground-truth graph if it exists or otherwise from the graph constructed using the kNN strategy. Best viewed in color.

function. For instance, for node classification problem, $\sigma(\cdot)$ 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.

We now discuss how to obtain the normalized adjacency matrix $\widetilde{\mathbf{A}}$. Our preliminary experiments showed that it is harmful to totally discard the initial graph structure when it is available. Previous works (Veličković et al. 2017; Jiang et al. 2019) inject the initial graph structure into the graph learning mechanism by performing masked attention, which might limits its graph learning ability. This is because there is no way for their methods to learn weights for those edges that do not exist in the initial graph, but carry useful topological information. With the assumption that the optimal graph structure is potentially a small shift from the initial graph structure, we combine the learned graph structure with the initial graph structure as follows,

$$\widetilde{\mathbf{A}} = \lambda \mathbf{L}_0 + (1-\lambda)\frac{\mathbf{A}_{ij}}{\sum_j \mathbf{A}_{ij}} \tag{9}$$

where $\mathbf{L}_0$ is the normalized adjacency matrix of the initial graph, defined as, $\mathbf{L}_0 = \mathbf{D}_0^{-1/2}\mathbf{A}_0\mathbf{D}_0^{-1/2}$, and $\mathbf{D}_0$ is its degree matrix. The adjacency matrix learned by Eqs. (1) and (2) is row normalized such that each row sums to 1. A 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 cosine similarity.

Iterative Method for Graph Learning Some previous works (Veličković et al. 2017) rely solely on raw node features to learn the graph structure based on some attention mechanism, which we think have some limitations since raw node features might not contain enough information to learn good graph structures. Our preliminary experiments showed that simply applying some attention function upon these raw node features does not help learn meaningful graphs (i.e., attention scores are kind of uniform). Even though we train the model jointly using the task-dependent prediction loss, we are limited by the fact that the similarity metric is computed based on the potentially inadequate raw node features.

To address the above limitation, we propose a Deep Iterative and Adaptive Learning framework for Graph Neural Networks (DIAL-GNN). A sketch of the DIAL-GNN framework is presented in Algorithm 1. Inputs and operations in squared brackets are optional. Specifically, besides computing the node similarity based on their raw fea-

Algorithm 1: DIAL-GNN

Input: $\mathbf{X}, \mathbf{y}[, \mathbf{A}_0]$
Parameters : $m, \varepsilon, \alpha, \beta, \gamma, \lambda, \delta, T, \eta[, k]$
Output: $\Theta, \widetilde{\mathbf{A}}^{(t)}, \hat{\mathbf{y}}$

$[\mathbf{A}_0 \leftarrow \text{kNN}(\mathbf{X}, k)]$ // Init. $\mathbf{A}_0$ to kNN graph if $\mathbf{A}_0$ is unavailable
$\mathbf{A}^{(0)}, \widetilde{\mathbf{A}}^{(0)} \leftarrow \{\mathbf{X}, \mathbf{A}_0\}$ using Eqs. (1), (2) and (9) // Learn the adj. matrix
$\mathbf{Z}^{(0)} \leftarrow \{\widetilde{\mathbf{A}}^{(0)}, \mathbf{X}\}$ using Eq. (6) // Compute node embeddings
$\mathcal{L}_{\text{pred}}^{(0)} \leftarrow \{\widetilde{\mathbf{A}}^{(0)}, \mathbf{Z}^{(0)}, \mathbf{y}\}$ using Eqs. (7) and (8) // Compute prediction loss
$\mathcal{L}_{\mathcal{G}}^{(0)} \leftarrow \{\mathbf{A}^{(0)}, \mathbf{X}\}$ using Eqs. (3)–(5) // Compute graph regularization loss
$\mathcal{L}^{(0)} \leftarrow \mathcal{L}_{\text{pred}}^{(0)} + \mathcal{L}_{\mathcal{G}}^{(0)}$ // Compute joint loss
$t \leftarrow 0$
while $(t == 0 \quad or \quad ||\mathbf{A}^{(t)} - \mathbf{A}^{(t-1)}||_F^2 > \delta||\mathbf{A}^{(0)}||_F^2) \quad and \quad t < T$ **do**
  $t \leftarrow t + 1$
  $\mathbf{A}^{(t)}, \widetilde{\mathbf{A}}^{(t)} \leftarrow \{\mathbf{Z}^{(t-1)}, \mathbf{A}_0\}$ using Eqs. (1), (2) and (9) // Refine the adj. matrix
  $\bar{\mathbf{A}}^{(t)} \leftarrow \{\widetilde{\mathbf{A}}^{(t)}, \widetilde{\mathbf{A}}^{(0)}\}$ using Eq. (10)
  $\mathbf{Z}^{(t)} \leftarrow \{\bar{\mathbf{A}}^{(t)}, \mathbf{X}\}$ using Eq. (6) // Refine node embeddings
  $\hat{\mathbf{y}} \leftarrow \{\bar{\mathbf{A}}^{(t)}, \mathbf{Z}^{(t)}\}$ using Eq. (7) // Compute task output
  $\mathcal{L}_{\text{pred}}^{(t)} \leftarrow \{\hat{\mathbf{y}}, \mathbf{y}\}$ using Eq. (8)
  $\mathcal{L}_{\mathcal{G}}^{(t)} \leftarrow \{\mathbf{A}^{(t)}, \mathbf{X}\}$ using Eqs. (3)–(5)
  $\mathcal{L}^{(t)} \leftarrow \mathcal{L}_{\text{pred}}^{(t)} + \mathcal{L}_{\mathcal{G}}^{(t)}$
end
$\mathcal{L} \leftarrow \mathcal{L}^{(0)} + \sum_{i=1}^{t} \mathcal{L}^{(i)}/t$
if *Training* **then**
  Back-propagate $\mathcal{L}$ to update model weights Θ
end

tures, we further introduce another learnable similarity metric function (Eq. (1)) that is rather computed based on the intermediate node embeddings, as demonstrated in Line 10. The aim is that the metric function defined on this node embedding space is able to learn topological information supplementary to the one learned solely based on the raw node features. In order to combine the advantages of both the raw node features and the node embeddings, we make the final learned graph structure as a linear combination of them,

$$\bar{\mathbf{A}}^{(t)} = \eta\widetilde{\mathbf{A}}^{(t)} + (1-\eta)\widetilde{\mathbf{A}}^{(0)} \tag{10}$$

where $\widetilde{\mathbf{A}}^{(t)}$ and $\widetilde{\mathbf{A}}^{(0)}$ are the two normalized adjacency matrices learned by Eq. (9) at the t-th iteration and the initialization step before the iterative loop, respectively.

Furthermore, as we can see from Line 10 to Line 12, the algorithm repeatedly refines the adjacency matrix $\widetilde{\mathbf{A}}^{(t)}$ with the updated node embeddings $\mathbf{Z}^{(t-1)}$, and in the meanwhile, refines the node embeddings $\mathbf{Z}^{(t)}$ with the updated adjacency matrix $\widetilde{\mathbf{A}}^{(t)}$. The iterative procedure dynamically stops when the learned adjacency matrix converges (with certain threshold δ) or the maximal number of iterations is reached (Line 8). 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. At each iteration, a joint loss combining both the task-dependent prediction loss and the graph regularization loss is computed (Line 16). After all iterations, the overall loss will be back-propagated through all previous iterations to update the model parameters (Line 20).

Formal Analysis

Convergence of the Iterative Learning Procedure

While it is challenging to theoretically prove the convergence of the proposed iterative learning procedure due to the arbitrary complexity of the involved learning model, here we want to conceptually understand why it works in practice. Fig. 2 shows the information flow of the learned adjacency matrix $\mathbf{A}$ and the intermediate 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 10), and $\mathbf{Z}^{(t)}$ is computed based on $\widetilde{\mathbf{A}}^{(t)}$ (Line 12) which is computed based on $\mathbf{A}^{(t)}$ (Eq. (9)). 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 $\delta_Z^{(t)}$.

If we assume that $\delta_Z^{(1)} < \delta_Z^{(0)}$, then we can expect that $\delta_A^{(2)} < \delta_A^{(1)}$ because conceptually more similar node embedding matrix (i.e., smaller δ_Z) is supposed to produce more similar adjacency matrix (i.e., smaller δ_A) given the fact that model parameters keep the same through iterations. Similarly, given that $\delta_A^{(2)} < \delta_A^{(1)}$, we can expect that $\delta_Z^{(2)} < \delta_Z^{(1)}$. Following this chain of reasoning, we can easily extend it to later iterations. In order to see why the assumption $\delta_Z^{(1)} < \delta_Z^{(0)}$ makes sense in practice, we need to recall the fact that $\delta_Z^{(0)}$ measures the difference between $\mathbf{Z}^{(0)}$ and $\mathbf{X}$, which is usually larger than the difference between $\mathbf{Z}^{(1)}$ and $\mathbf{Z}^{(0)}$, namely $\delta_Z^{(1)}$. We will empirically examine the convergence property of the iterative learning procedure in the experimental section.

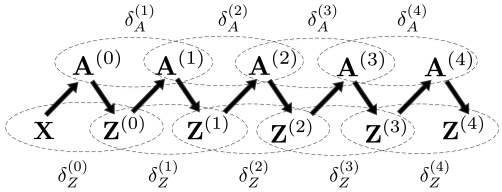

Figure 2: Information flow of the proposed iterative learning procedure.

Model Complexity

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 $\mathcal{O}(n^2d + ndh)$, computing task output costs $\mathcal{O}(n^2h)$, and computing the total loss costs $\mathcal{O}(n^2d)$. 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 complexity is $\mathcal{O}(Tdn^2)$.

Experiments

In this section, we conducted a series of experiments to verify the effectiveness of the proposed model and assess the impact of different model components. The implementation of the model will be made publicly available at https://github.com/hugochan/IDGL soon. The details on model settings are provided in the appendix.

Datasets and Setup

The benchmarks used in our experiments include two network benchmarks, three data point benchmarks and two text benchmarks. Cora and Citeseer are two commonly used network benchmarks for evaluating graph-based learning algorithms (Sen et al. 2008). The input features are bag of words and the task is node classification. In addition to Cora and Citeseer where the graph topology is available, we evaluate DIAL-GNN on three data point benchmarks (i.e., Wine, Breast Cancer (Cancer) and Digits from the UCI machine learning repository (Dua and Graff 2017)). The task is also node classification. Finally, to demonstrate the effectiveness of DIAL-GNN on inductive learning problems, we conduct document classification and regression tasks on the 20Newsgroups data (20News) and the movie review data (MRD) (Pang and Lee 2004), respectively. In this setting, we regard each document as a graph containing each word as a node.

For Cora and Citeseer, we follow the experimental setup of previous works (Kipf and Welling 2016; Veličković et al.

Table 1: Test accuracy ($\pm$ standard deviation) in percentage on various classification datasets in the transductive setting.

Methods	Cora	Citeseer	Wine	Cancer	Digits
RBF SVM	59.7 (0.0)	60.2 (0.0)	94.1 (2.9)	91.7 (3.1)	86.9 (3.2)
SemiEmb	63.1 (0.1)	68.1 (0.1)	91.9 (0.1)	89.7 (0.1)	90.9 (0.1)
LDS	84.1 (0.4)	75.0 (0.4)	97.3 (0.4)	94.4 (1.9)	92.5 (0.7)
GCN	81.0 (0.2)	70.9 (0.3)	—	—	—
GAT	82.5 (0.4)	70.9 (0.4)	—	—	—
kNN-GCN	—	—	95.9 (0.9)	94.7 (1.2)	89.5 (1.3)
LDS*	83.9 (0.6)	**74.8 (0.3)**	96.9 (1.4)	93.4 (2.4)	90.8 (2.5)
DIAL-GNN	**84.5 (0.3)**	74.1 (0.2)	**97.8 (0.6)**	**95.1 (1.0)**	**93.1 (0.5)**

Table 2: Test scores ($\pm$ standard deviation) in percentage on classification (accuracy) and regression (R^2) datasets in the inductive setting.

Methods	20News	MRD
BiLSTM	80.0 (0.4)	53.1 (1.4)
kNN-GCN	81.3 (0.6)	60.1 (1.5)
DIAL-GNN	**83.6 (0.4)**	**63.7 (1.8)**

Table 3: Ablation study on various classification datasets.

Methods	Cora	Wine	20News
DIAL-GNN	**84.5 (0.3)**	**97.8 (0.6)**	**83.6 (0.4)**
w/o graph reg.	84.3 (0.4)	97.3 (0.8)	83.4 (0.5)
w/o IL	83.5 (0.6)	97.2 (0.8)	83.0 (0.4)

2017; Franceschi et al. 2019). For Wine, Cancer and Digits, we follow the experimental setup of (Franceschi et al. 2019). For 20News, we randomly select 30% examples from the training data as the development set. For MRD, we split the data to train/dev/test sets using a 60%/20%/20% split. The reported results are averaged over 5 runs with different random seeds. Please refer to the appendix for data statistics.

Baselines

Our main baseline in the transductive setting is LDS. Similar to our work, LDS also jointly learns the graph structure and the parameters of GNNs. However, LDS is incapable of handling inductive learning problems since it aims at directly optimizing the discrete probability distribution on the edges of the underlying graph, which makes it unable to handle unseen nodes/graphs in the testing phase. The experimental results of several semi-supervised embedding (SemiEmb) (Weston et al. 2012)) and supervised learning (support vector machines (RBF SVM)) baselines are reported in the LDS paper. For the sake of completeness, we directly copy their results here. For ease of comparison, we also copy the reported results of LDS even though we rerun the experiments of LDS using the official code released by the authors.

In addition, for Cora and Citeseer, we include GCN (Kipf and Welling 2016) and GAT (Veličković et al. 2017) as baselines. In order to evaluate the robustness of DIAL-GNN to noisy graphs, we also compare DIAL-GNN with GCN on graphs with edge deletions or additions. For data point benchmarks where the graph topology is not available, we conceive a kNN-GCN baseline where a kNN affinity graph on the data set is first constructed as a preprocessing step before applying a GCN. For 20News and MRD in the inductive setting, we compare DIAL-GNN with a BiLSTM (Hochreiter and Schmidhuber 1997) baseline and kNN-GCN.

Results and Analysis

The results of transductive and inductive experiments are shown in Table 1 and Table 2. First of all, we can see that DIAL-GNN outperforms all baseline methods in 6 out of 7 benchmarks, which demonstrates the effectiveness of DIAL-GNN. We can clearly see that DIAL-GNN can greatly help the node classification task even when the graph topology is given. When the graph topology is not given, compared to kNN-GCN, DIAL-GNN consistently achieves much better results on all datasets, which shows the power of jointly learning graph structures and GNN parameters. Compared to LDS, DIAL-GNN achieves better performance in 4 out of 5 benchmarks. The good performance on 20News and MRD verifies the capability of DIAL-GNN on inductive learning problems.

We perform an ablation study to assess the impact of different model components. As shown in Table 3, we can see a significant performance drop consistently on all datasets (e.g., 3.1% on Citeseer) by turning off the iterative learning component, which demonstrates the effectiveness of the proposed iterative learning framework for the graph learning problem. We can also see the benefits of jointly training the model with the graph regularization loss.

To evaluate the robustness of DIAL-GNN on noisy graphs, we construct graphs with random edge deletions or additions. Specifically, we randomly remove or add 25%, 50% and 75% of the edges in the original graphs. The results on the edge deletion graphs and edge addition graphs are shown in Fig. 3 and Fig. 4, respectively. As we can clearly see, compared to GCN, DIAL-GNN achieves better results in all scenarios and is much more robust to noisy graphs. While GCN completely fails in the edge addition scenario, DIAL-GNN is still able to perform reasonably well. We conjecture this is because Eq. (9) is formulated in a form of skip-connection, by lowering the value of λ, we enforce the model to rely less on the initial noisy graph that contains too much additive random noise.

In Fig. 5, 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 ac-

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

Benchmarks	Cora	Citeseer	Wine	Cancer	Digits
GCN	3 (1)	5 (1)	—	—	—
GAT	26 (5)	28 (5)	—	—	—
LDS	390 (82)	585 (181)	33 (15)	25 (6)	72 (35)
DIAL-GNN	237 (21)	563 (100)	20 (7)	21 (11)	65 (12)
DIAL-GNN w/o IL	49 (8)	61 (15)	3 (2)	3 (1)	2 (1)

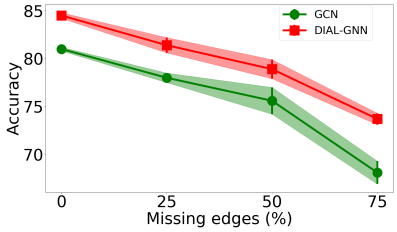

Figure 3: Test accuracy ($\pm$ standard deviation) in percentage for the edge deletion scenario on Cora.

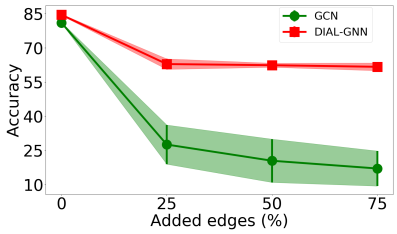

Figure 4: Test accuracy ($\pm$ standard deviation) in percentage for the edge addition scenario on Cora.

curacy converge quickly through iterations. This empirically verifies the analysis we made on the convergence property of the iterative learning procedure.

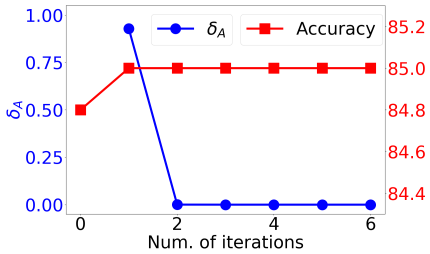

Figure 5: Evolution of the learned adjacency matrix and test accuracy (in %) through iterations in the iterative learning procedure.

There are two natural ways of designing the stopping strategy for iterative learning methods. We can either use a fixed number of iterations, or dynamically determine if the learning procedure already converges or not based on some stopping criterion. In Fig. 6, we empirically compare the effectiveness of the above two strategies. We run DIAL-GNN on Cora (left) and Citeseer (right) using different stopping strategies with 5 runs, and report the average accuracy. As we can see, dynamically adjusting the number of iterations using the stopping criterion works better in practice.

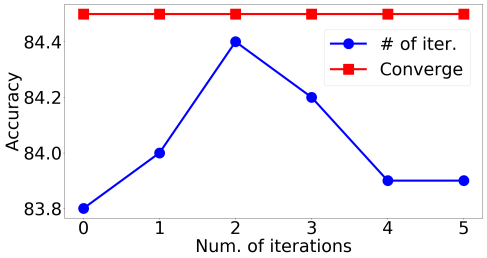

Figure 6: Performance comparison (i.e., test accuracy in %) of two different stopping strategies: i) using a fixed number of iterations (blue line), and ii) using a stopping criterion to dynamically determine the convergence (red line).

Timing

Finally, we compare the training efficiency of DIAL-GNN, LDS and other classic GNNs (e.g., GCN and GAT) on various benchmarks. All experiments are conducted on the same machine which has an Intel i7-2700K CPU, an Nvidia Titan Xp GPU and 16GB RAM, and are repeated 5 times with different random seeds. Results are shown in Table 4. As we can see, both DIAL-GNN and LDS are slower than GCN and GAT, which is as expected since GCN and GAT do not need to learn graph structures simultaneously. DIAL-GNN is consistently faster than LDS, but in general, they are comparable. We also find that the iterative learning part is the most time consuming in DIAL-GNN.

Conclusion

In this paper, we proposed a Deep Iterative and Adaptive Learning framework for Graph Neural Networks (DIAL-GNN) for jointly learning the graph structure and graph embeddings by optimizing a joint loss combining both task prediction loss and graph regularization loss. The proposed method is able to iteratively search for hidden graph structures that better help the downstream prediction task. Our extensive experiments demonstrate that the proposed DIAL-GNN model can consistently outperform or match state-of-the-art baselines on various datasets. We leave how to design more effective and scalable metric functions as future work.

Acknowledgments This work is supported by IBM Research AI through the IBM AI Horizons Network. We thank the anonymous reviewers for their helpful feedback.

References

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

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

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

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

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

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

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

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

Gao, Y.; Wu, L.; Homayoun, H.; and Zhao, L. 2019. Dyngraph2seq: Dynamic-graph-to-sequence interpretable learning for health stage prediction in online health forums. *arXiv preprint arXiv:1908.08497*.

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

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

Hoshen, Y. 2017. Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, 2701–2711.

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

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

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

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

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

Kipf, T.; Fetaya, E.; Wang, K.-C.; Welling, M.; and Zemel, R. 2018. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*.

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

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

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

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

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

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

Shuman, D. I.; Narang, S. K.; Frossard, P.; Ortega, A.; and Vandergheynst, P. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine* 30(3):83–98.

Sukhbaatar, S.; Fergus, R.; et al. 2016. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, 2244–2252.

Van Steenkiste, S.; Chang, M.; Greff, K.; and Schmidhuber, J. 2018. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*.

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

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

Weston, J.; Ratle, F.; Mobahi, H.; and Collobert, R. 2012. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*. Springer. 639–655.

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

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

Xu, K.; Wu, L.; Wang, Z.; Yu, M.; Chen, L.; and Sheinin, V. 2018c. Sql-to-text generation with graph-to-sequence model. *arXiv preprint arXiv:1809.05255*.

Data Statistics

The benchmarks used in our experiments include two network benchmarks, three data point benchmarks and two text benchmarks. Below we show the brief data statistics.

Table 5: Data statistics.

Benchmarks	Train/Dev/Test	Task	Setting
Cora	140/500/1,000	node clf	transductive
Citeseer	120/500/1,000	node clf	transductive
Wine	10/20/158	node clf	transductive
Cancer	10/20/539	node clf	transductive
Digits	50/100/1,647	node clf	transductive
20News	7,919/3,395/7,532	graph clf	inductive
MRD	3,003/1,001/1,002	graph reg	inductive

Model Settings

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 maximal 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. We use Adam (Kingma and Ba 2014) 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. Table 6 shows the hyperparameters associated to DIAL-GNN for all benchmarks. All hyperparameters are tuned on the development set.

Table 6: Hyperparameter associated to DIAL-GNN 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	4e-5	10
Citeseer	0.6	0.5	0.4	0.0	0.2	–	0.3	1	1e-3	10
Wine	0.8	0.7	0.1	0.1	0.3	20	0.75	1	1e-3	10
Cancer	0.25	0.1	0.4	0.2	0.1	40	0.9	1	1e-3	10
Digits	0.4	0.1	0.4	0.1	0.0	24	0.65	8	1e-4	10
20News	0.1	0.4	0.5	0.01	0.3	950	0.3	12	8e-3	10
MRD	0.5	0.9	0.2	0.0	0.1	350	0.4	5	4e-2	10