Combining Graph Convolutional Neural Networks and Label Propagation

HONGWEI WANG and JURE LESKOVEC, Stanford University, United States

Label Propagation Algorithm (LPA) and Graph Convolutional Neural Networks (GCN) are both message passing algorithms on graphs. Both solve the task of node classification, but LPA propagates node label information across the edges of the graph, while GCN propagates and transforms node feature information. However, while conceptually similar, theoretical relationship between LPA and GCN has not yet been systematically investigated. Moreover, it is unclear how LPA and GCN can be combined under a unified framework to improve the performance. Here we study the relationship between LPA and GCN in terms of *feature/label influence*, in which we characterize how much the initial feature/label of one node influences the final feature/label of another node in GCN/LPA. Based on our theoretical analysis, we propose an end-to-end model that combines GCN and LPA. In our unified model, edge weights are learnable, and the LPA serves as regularization to assist the GCN in learning proper edge weights that lead to improved performance. Our model can also be seen as learning the weights of edges based on node labels, which is more direct and efficient than existing feature-based attention models or topology-based diffusion models. In a number of experiments for semi-supervised node classification and knowledge-graph-aware recommendation, our model shows superiority over state-of-the-art baselines.

CCS Concepts: • **Computing methodologies → Neural networks**; **Semi-supervised learning settings;**

Additional Key Words and Phrases: Graph convolutional neural networks; label propagation algorithm; semi-supervised learning

ACM Reference format:
Hongwei Wang and Jure Leskovec. 2021. Combining Graph Convolutional Neural Networks and Label Propagation. *ACM Trans. Inf. Syst.* 40, 4, Article 73 (November 2021), 27 pages.
https://doi.org/10.1145/3490478

1 INTRODUCTION

Consider the problem of node classification in a graph, where the goal is to learn a mapping $\mathcal{M} : V \rightarrow L$ from node set V to label set L. A solution to this problem is widely applicable to

Some of the experimental results come from our previous work at the 2019 ACM International Conference on Knowledge Discovery and Data Mining (KDD 2019) [35].

This research has been supported in part by DARPA, ARO, NSF, NIH, Stanford Data Science Initiative, Wu Tsai Neurosciences Institute, Chan Zuckerberg Biohub, Amazon, JPMorgan Chase, Docomo, Hitachi, Intel, JD.com, KDDI, NVIDIA, Dell, Toshiba, Visa, and UnitedHealth Group.

Authors' addresses: H. Wang and J. Leskovec, Computer Science Department, Stanford University, Stanford, CA 94305, United States; emails: {hongweiw, jure}@cs.stanford.edu.

1046-8188/2021/11-ART73 $15.00
https://doi.org/10.1145/3490478

various scenarios, e.g., inferring income of users in a social network or classifying scientific articles in a citation network. Different from a generic machine learning problem where samples are independent from each other, nodes are connected by edges in the graph, which provide additional information and require more delicate modeling. To capture the graph information, researchers have mainly designed models on the assumption that labels/features are correlated over the edges of th 4e graph. In particular, on the label side L, node labels are propagated and aggregated along edges in the graph, which is known as **Label Propagation Algorithm (LPA)** [5, 10, 21, 33, 45, 47, 49]; On the feature side V, node features are propagated along edges and transformed through neural network layers, which is known as **Graph Convolutional Neural Networks (GCN)**[1] [6, 12, 18, 20, 23, 39, 40].

GCN and LPA are related in that they propagate features and labels on the two sides of the mapping $\mathcal{M}$, respectively. Prior work [19] has shown the relationship between GCN and LPA in terms of low-pass graph filtering. However, it is unclear how the discovered relationship benefits node classification. Specifically, can GCN and LPA be combined to develop a more accurate model for node classification in graphs? Here we study the theoretical relationship between GCN and LPA from the viewpoint of *feature/label influence*, where we quantify how much the initial label of node v_b influences the output label of node v_a in LPA by studying the gradient of node v_b with respect to node v_a, and how much the initial feature of node v_b influences the output feature of node v_a in GCN by studying the Jacobian of node v_b with respect to node v_a. We also prove the quantitative relationship between feature influence and label influence by showing that the label influence of v_b on v_a equals the cumulative discounted feature influence of v_b on v_a in expectation (see Theorem 1).

Based on the theoretical analysis, we propose a unified model GCN-LPA for node classification. We show that the key to improving the performance of GCN is to enable nodes of the same class to connect more strongly with each other by making edge weights/strengths trainable. Then we prove that increasing the strength of edges between the nodes of the same class is equivalent to increasing the accuracy of LPA's predictions (see Theorem 2). Therefore, we can first learn the optimal edge weights by minimizing the loss of predictions in LPA, then plug the optimal edge weights into a GCN to learn node representations. In GCN-LPA, we further combine the above two steps together and train the whole model in an end-to-end fashion, where the LPA part serves as regularization to assist the GCN part in learning proper edge weights that benefit the separation of different node classes.

It is worth noticing that GCN-LPA can also be seen as learning the weights for edges based on *node label information*, which requires less handcrafting and is more task-oriented than existing attention models that learn edge weights based on *node feature similarity* [22, 31, 32, 44] or diffusion models that learn adjacency matrix based on *graph topology* [1, 13, 14, 38].

We conduct extensive experiments in two real-world tasks: semi-supervised node classification and knowledge-graph-aware recommendation. Empirical results demonstrate that our unified model outperforms state-of-the-art methods by a large margin. The experimental results also show that combining GCN and LPA together is able to learn more informative edge weights thereby leading to better performance.

Our contribution in this article are listed as follows:

- We systematically study the theoretical relationship between GCN and LPA in terms of feature/label influence.

[1]There are methods in statistical relational learning [25] also using feature propagation/diffusion techniques. In this work, we focus on GCN, but the analysis and the proposed model can be easily generalized to other feature diffusion methods.

- Based on the theoretical analysis, we propose an end-to-end model that combines GCN and LPA together under a unified framework.
- We conduct extensive experiments on real-world graphs, and the results demonstrate the efficacy of our proposed model in graph-related tasks.

2 PROBLEM FORMULATION AND PRELIMINARIES

In this section, we first formulate the problem then briefly introduce LPA and GCN.

2.1 Problem Formulation

Consider a graph $G = (V, A, X, Y)$, in which $V = \{v_1, \ldots, v_n\}$ is the set of nodes, $A \in \mathbb{R}^{n\times n}$ is the adjacency matrix, X is the feature matrix of nodes, and Y is labels of nodes. a_{ij} (the ijth entry of A) is the weight of the edge connecting v_i and v_j. $N(v)$ denotes the set of first-order neighbors of node v in graph G. Each node v_i has a feature vector $\mathbf{x}_i$, which is the ith row of X, while only the first m nodes ($m \ll n$) have labels $y_1, \ldots, y_m$ from a label set $L = \{1, \ldots, c\}$. The goal is to learn a mapping $\mathcal{M} : V \to L$ and predict labels of unlabeled nodes.

2.2 Label Propagation Algorithm

LPA [49] assumes that two connected nodes are likely to have the same label, and thus it propagates labels iteratively along the edges. Let $Y^{(k)} = [y_1^{(k)}, \ldots, y_n^{(k)}]^\top \in \mathbb{R}^{n\times c}$ be the soft label matrix in iteration $k > 0$, in which the ith row $y_i^{(k)\top}$ denotes the predicted label distribution for node v_i in iteration k. When $k = 0$, the initial label matrix $Y^{(0)} = [y_1^{(0)}, \ldots, y_n^{(0)}]^\top$ consists of one-hot label indicator vectors $y_i^{(0)}$ for $i = 1, \ldots, m$ (i.e., labeled nodes) or zero vectors otherwise (i.e., unlabeled nodes). Then LPA in iteration k is formulated as the following two steps:

$$Y^{(k+1)} = \tilde{A}\, Y^{(k)}, \tag{1}$$

$$y_i^{(k+1)} = y_i^{(0)},\ \forall\, i \le m. \tag{2}$$

In the above equations, $\tilde{A}$ is the normalized adjacency matrix, which can be the random walk transition matrix $\tilde{A}_{rw} = D^{-1}A$ or the symmetric transition matrix $\tilde{A}_{sym} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, where D is the diagonal degree matrix for A with entries $d_{ii} = \sum_j a_{ij}$. Without loss of generality, we use $\tilde{A} = \tilde{A}_{rw}$ in this work. In Equation (1), all nodes propagate labels to their neighbors according to normalized edge weights. Then, in Equation (2), labels of all labeled nodes are reset to their initial values, because LPA wants to persist labels of nodes that are labeled, so that unlabeled nodes do not overpower the labeled ones as the initial labels would otherwise fade away.

2.3 Graph Convolutional Neural Networks

GCN [12] is a multi-layer feedforward neural network that propagates and transforms node features across the graph. The feature propagation scheme of GCN in layer k is as follows:

$$X^{(k+1)} = \sigma(\tilde{A}X^{(k)}W^{(k)}), \tag{3}$$

where $W^{(k)}$ is trainable weight matrix in the kth layer, $\sigma(\cdot)$ is an activation function, and $X^{(k)} = [\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_n^{(k)}]^\top$ are the kth layer node representations with $X^{(0)} = X$. By setting the dimension of the last layer to the number of classes c, the last layer can be seen as (unnormalized) label distribution predicted for a given node. The whole model can thus be optimized by minimizing the discrepancy between predicted node label distributions and ground-truth labels Y.

Notice similarity between Equations (1) and (3). Next we shall study and uncover the relationship between the two equations.

3 FEATURE INFLUENCE AND LABEL INFLUENCE

Consider two nodes v_a and v_b in a graph. Inspired by Reference [15] and Reference [40], we study the relationship between GCN and LPA in terms of influence, i.e., how the output feature/label of v_a will change if the initial feature/label of v_b is varied slightly. Technically, the feature/label influence is measured by the Jacobian/gradient of the output feature/label of v_a with respect to the initial feature/label of v_b. Denote $\mathbf{x}_a^{(k)}$ as the kth layer representation vector of v_a in GCN, and $\mathbf{x}_b$ as the initial feature vector of v_b. We quantify the feature influence of v_b on v_a as follows:

Definition 1 (Feature Influence). The feature influence of node v_b on node v_a after k layers of GCN is the L1-norm of the expected Jacobian matrix $\partial \mathbf{x}_a^{(k)} / \partial \mathbf{x}_b$:

$$I_f(v_a, v_b; k) = \left\| \mathbb{E}_{W^{(\cdot)}} \left[\frac{\partial \mathbf{x}_a^{(k)}}{\partial \mathbf{x}_b} \right] \right\|_1, \tag{4}$$

where the expectation is taken on transformation matrices $W^{(0)}, \ldots, W^{(k-1)}$. The normalized feature influence is then defined as

$$\tilde{I}_f(v_a, v_b; k) = \frac{I_f(v_a, v_b; k)}{\sum_{v_i \in V} I_f(v_a, v_i; k)}. \tag{5}$$

We also consider the label influence of node v_b on node v_a in LPA (this implies that v_a is unlabeled and v_b is labeled). Since different label dimensions of $y_i^{(\cdot)}$ do not interact with each other in LPA, we assume that all y_i and $y_i^{(\cdot)}$ are scalars within range $[0, 1]$ (i.e., this is a binary classification task) for simplicity. Label influence is defined as follows:

Definition 2 (Label Influence). The label influence of labeled node v_b on unlabeled node v_a after k iterations of LPA is the gradient of $y_a^{(k)}$ with respect to y_b:

$$I_l(v_a, v_b; k) = \frac{\partial y_a^{(k)}}{\partial y_b}. \tag{6}$$

The following theorem shows the relationship between feature influence and label influence:

Theorem 1 (Relationship Between Label Influence and Feature Influence). *Assume the activation function used in GCN is* $ReLU(x) = \max(x, 0)$. *Denote v_a as an unlabeled node, v_b as a labeled node, and β as the fraction of unlabeled nodes. Then the label influence of v_b on v_a after k iterations of LPA equals, in expectation, to the cumulative normalized feature influence of v_b on v_a after k layers of GCN:*

$$\mathbb{E}_{W^{(\cdot)}} \left[I_l(v_a, v_b; k) \right] = \sum_{j=1}^{k} \beta^j \tilde{I}_f(v_a, v_b; j). \tag{7}$$

Proof of Theorem 1 is in Appendix A. Intuitively, Theorem 1 shows that *if v_b has high label influence on v_a, then the initial feature vector of v_b will also affect the output feature vector of v_a to a large extent.* Theorem 1 provides the theoretical guideline for designing our unified model in the next section.

4 THE UNIFIED MODEL: GCN-LPA

Before introducing the proposed model, we rethink the GCN method and see what an ideal set of node representations should be like. Since we aim to classify nodes, the perfect node representation would be such that nodes with the same label are embedded closely together, which would give a large separation between different classes. Intuitively, the key to achieve this goal is to enable nodes within the same class to connect more strongly with each other, so that they are pushed together by GCN (more discussion is presented in Section 5). We can therefore make edge strengths/weights trainable, then learn to increase the *intra-class feature influence*:

$$\sum_{i \in L} \sum_{\substack{v_a, v_b: \\ y_a = i, y_b = i}} \tilde{I}_f(v_a, v_b) \tag{8}$$

(L is the label set) by adjusting edge weights. However, this requires operating on Jacobian matrices with the size of $d^{(0)} \times d^{(K)}$ ($d^{(0)}$ and $d^{(K)}$ are the dimensions of input and output in GCN, respectively), which is impractical if initial node features are high dimensional. Fortunately, we can turn to optimizing the *intra-class label influence* instead, i.e.,

$$\sum_{i \in L} \sum_{\substack{v_a, v_b: \\ y_a = i, y_b = i}} I_l(v_a, v_b), \tag{9}$$

according to Theorem 1. Note that

$$\sum_{i \in L} \sum_{\substack{v_a, v_b: \\ y_a = i, y_b = i}} I_l(v_a, v_b) = \sum_{v_a} \sum_{\substack{v_b: \\ y_b = y_a}} I_l(v_a, v_b). \tag{10}$$

We further show, by the following theorem, that the term $\sum_{v_b : y_b = y_a} I_l(v_a, v_b)$ (the total intra-class label influence on a given node v_a) is proportional to the probability that v_a is classified correctly by LPA:

Theorem 2 (Relationship Between Label Influence and LPA's Prediction). *Consider a given node v_a and its label y_a. If we treat node v_a as unlabeled, then the total label influence of nodes with label y_a on node v_a is proportional to the probability that node v_a is classified as y_a by LPA:*

$$\sum_{v_b : y_b = y_a} I_l(v_a, v_b; k) \propto \Pr\left(\hat{y}_a^{lpa} = y_a\right), \tag{11}$$

where $\hat{y}_a^{lpa}$ is the predicted label of v_a using a k-iteration LPA.

Proof of Theorem 2 is in Appendix B. Theorem 2 indicates that, *if edge weights $\{a_{ij}\}$ maximize the probability that v_a is correctly classified by LPA, then they also maximize the intra-class label influence on node v_a*. We can therefore first learn the optimal edge weights A^* by minimizing the loss of predicted labels by LPA:[2]

$$A^* = \arg\min_A L_{lpa}(A) = \arg\min_A \frac{1}{m} \sum_{v_a : a \le m} J\left(\hat{y}_a^{lpa}, y_a\right), \tag{12}$$

where J is the cross-entropy loss and $\hat{y}_a^{lpa}$ and y_a are the predicted label distribution of v_a using LPA and the true one-hot label of v_a, respectively. $a \le m$ means v_a is labeled. The optimal A^* maximizes the probability that each node is correctly labeled by LPA (according to the definition of A^* in Equation (12)), thus also maximizes the intra-class label influence (according to Theorem 2) and intra-class feature influence (according to Theorem 1). Since A^* increases the connection strength of nodes within each class (according to Definition 1), it is expected to improve the performance of GCN compared with the original adjacency matrix A. Therefore, we can plug A^* into GCN to predict labels:

$$X^{(k+1)} = \sigma\left(A^* X^{(k)} W^{(k)}\right), \ k = 0, 1, \ldots, K-1. \tag{13}$$

[2]Here the optimal edge weights A^* share the same topology as the original graph G, i.e., we do not add or remove edges from G but only learning the weights of existing edges. See the end of this section for more discussion.

We use $\hat{y}_a^{gcn}$, the ath row of $X^{(K)}$, to denote the predicted label distribution of v_a using the GCN specified in Equation (13). Then the optimal transformation matrices in the GCN can be learned by minimizing the loss of predicted labels by GCN:

$$W^* = \arg\min_W L_{gcn}(W, A^*) = \arg\min_W \frac{1}{m} \sum_{v_a : a \le m} J\left(\hat{y}_a^{gcn}, y_a\right), \tag{14}$$

It is more elegant (and empirically better) to combine the above two steps together into a multi-objective optimization problem and train the whole model in an end-to-end fashion:

$$W^*, A^* = \arg\min_{W,A} \; L_{gcn}(W, A) + \lambda L_{lpa}(A), \tag{15}$$

where λ is the balancing hyper-parameter. In this way, $L_{lpa}(A)$ serves as a regularization term that assists the learning of edge weights A, since it is hard for GCN to learn both W and A simultaneously due to overfitting. The proposed GCN-LPA approach can also be seen as learning the importance of edges that can be used to reconstruct node labels accurately by LPA, then transferring this knowledge from label space to feature space for GCN.

It is also worth noticing how the optimal A^* is configured. The principle here is that we do not modify the basic structure of the original graph (i.e., not adding or removing edges) but only adjusting weights of existing edges. This is equivalent to learning a positive mask matrix M for the adjacency matrix A and taking the Hadamard product $M \circ A = A^*$. In general, we have two options to design the mask matrix M:

- Each element M_{ij} can be set as a free variable during training. This applies to the case where no node feature or edge feature is available. However, the drawbacks of this option are that, it can only work in transductive setting while the trained model (learned edge weights) cannot be used for new graphs, and the model may be extremely large for large graphs, since the number of model parameters increases linearly with the number of edges.
- Each element M_{ij} can be set as a function of features of two endpoints and/or the edge, for example, $M_{ij} = \kappa(\mathbf{x}_i^\top \mathbf{H} \mathbf{x}_j)$, where $\mathbf{H}$ is a learnable kernel matrix for measuring node feature similarity and $\kappa(\cdot)$ is a mapping from $\mathbb{R}$ to $\mathbb{R}^+$ such as softplus or softmax, or $M_{ij} = \kappa(\mathbf{h}^\top \mathbf{x}_{ij})$ where $\mathbf{h}$ is a trainable vector to transform edge features to weights. In this way, the model does not use any node or edge identity, and the learned $\mathbf{H}$ or $\mathbf{h}$ can be applied to new graphs in inductive settings. Moreover, the model size is also independent with the graph size.

In our experiments, we use the first option in semi-supervised node classification task (Section 6.1) and the second option in knowledge-graph-aware recommendation task (Section 6.2).

5 ANALYSIS OF GCN-LPA MODEL BEHAVIOR

In this section, we show benefits of our unified model compared with GCN by analyzing properties of embeddings produced by the two models. We first analyze the update rule of GCN for node v_i,

$$\mathbf{x}_i^{(k+1)} = \sigma\left(\sum_{v_j \in N(v_i)} \tilde{a}_{ij} \mathbf{x}_j^{(k)} W^{(k)} \right), \tag{16}$$

where $\tilde{a}_{ij} = a_{ij}/d_{ii}$ is the normalized weight of edge (j, i). This formula can be decomposed into the following two steps:

- In *aggregation* step, we calculate the aggregated representation $\mathbf{h}_i^{(k)}$ of all neighborhoods $N(v_i)$:
$$\mathbf{h}_i^{(k)} = \sum_{v_j \in N(v_i)} \tilde{a}_{ij} \mathbf{x}_j^{(k)}. \tag{17}$$
- In *transformation* step, the aggregated representation $\mathbf{h}_i^{(k)}$ is mapped to a new space by a transformation matrix and nonlinear function:
$$\mathbf{x}_i^{(k+1)} = \sigma\left(\mathbf{h}_i^{(k)} W^{(k)}\right). \tag{18}$$

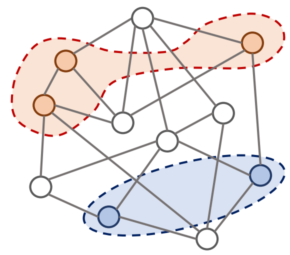
We show by the following theorem that the aggregation step reduces the overall distance in the embedding space between the nodes that are connected in the graph:

Theorem 3 (Shrinking Property in GCN). *If we define*
$$D(\mathbf{x}) = \frac{1}{2} \sum_{v_i, v_j} \tilde{a}_{ij} \left\| \mathbf{x}_i - \mathbf{x}_j \right\|_2^2 \tag{19}$$
as a distance metric over node embeddings $\mathbf{x}$, *then we have*
$$D\left(\mathbf{h}^{(k)}\right) \leq D\left(\mathbf{x}^{(k)}\right). \tag{20}$$

Proof of Theorem 3 is in Appendix C. Theorem 3 indicates that *the overall distance among connected nodes is reduced after taking one aggregation step*, which implies that connected components in the graph "shrink" and nodes within each connected component get closer to each other in the embedding space. In an ideal case where edges only connect nodes with the same label, the aggregation step will push nodes within the same class together, which greatly benefits the transformation step that acts like using a hyperplane $W^{(k)}$ for classification. However, two connected nodes may have different labels. These "noisy" edges will impede the formation of clusters and make the inter-class boundary less clear.

Fortunately, in GCN-LPA, edge weights are learned by minimizing the difference between ground-truth labels and predicted labels using LPA. As we know that LPA predicts the label of a given node by propagating and aggregating the labels of nearby nodes to the given node. Therefore, to force the predicted label of a node to approach its ground truth, our model will learn to increase the weight/bandwidth of possible paths that connect the given node and its nearby nodes with the same label, so that their labels can "flow" easily along these paths to the given node (the given node's own label is masked when predicting itself, so the model cannot learn to "cheat" by increasing the weight of the self-loop edge). In this way, GCN-LPA is able to identify potential intra-class edges. Note that labeled nodes and unlabeled nodes are usually mixed in a graph, so the intra-class paths between labeled nodes will also connect many unlabeled nodes (see Figure 1 for an illustrating example). Increasing the weight of these intra-class paths can therefore assist learning clustering structures for both labeled and unlabeled nodes, and improve the classification accuracy for unlabeled nodes.

To empirically justify our claim, we apply a two-layer untrained GCN with randomly initialized transformation matrices to a subgraph of Cora dataset (see Section 6.1.1 for detailed description on Cora), which contains randomly selected 50 nodes with label 0 and randomly selected 50 nodes with label 1, as well as all edges between these nodes (grey lines). We then increase the weights of intra-class edges by 10 times to simulate GCN-LPA. The initial node features are set as default in Cora. We find that GCN works well on this network (Figure 2(a)), but GCN-LPA performs even better than GCN, because the node embeddings are almost linearly separable as shown in Figure 2(b). To further justify our claim, we randomly add 50 "noisy" inter-class edges to the original network, from which we observe that GCN is misled by noise and mixes nodes of two classes


(a) A graph with two classes of nodes

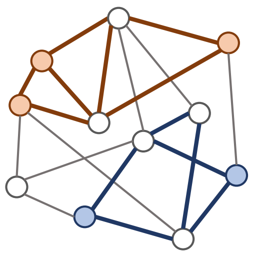
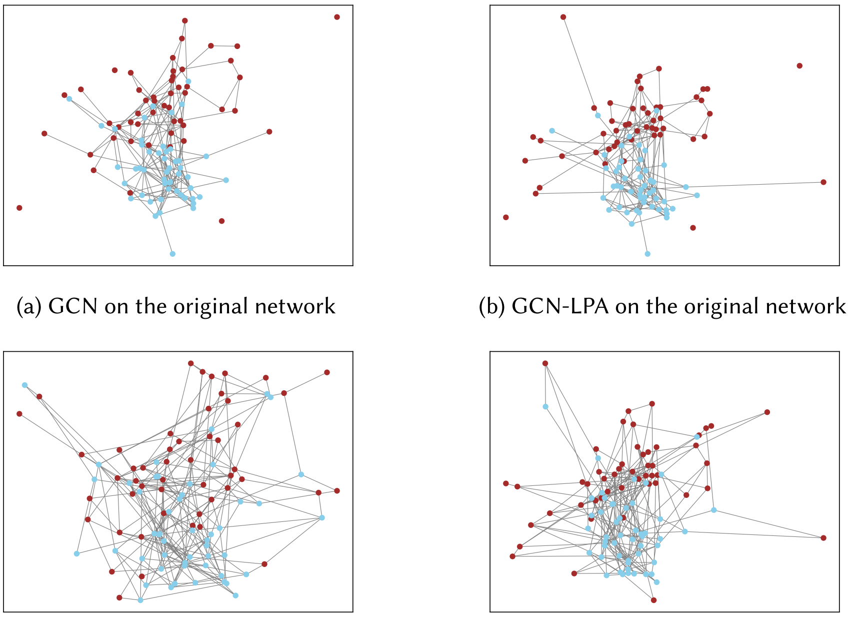

(b) Potential intra-class edges (bold links)

Fig. 1. A graph with two classes of nodes, while white nodes are unlabeled (Figure 1(a)). To classify nodes, our model will increase the connecting strength among nodes within the same class, thereby increasing their feature/label influence on each other. In this way, our model is able to identify potential intra-class edges (bold links in Figure 1(b)) and strengthen their weights.


(a) GCN on the original network

(b) GCN-LPA on the original network

(c) GCN on the noisy network

(d) GCN-LPA on the noisy network

Fig. 2. Node embeddings of a subgraph of Cora trained on a node classification task (red vs. blue). Node coordinates in Figures 2(a)–2(d) are the embedding coordinates. Notice that GCN does not produce linearly separable embeddings (Figure 2(a) vs. Figure 2(b)), while GCN-LPA performs much better even in the presence of noisy edges (Figure 2(c) vs. Figure 2(d)). Additional visualizations are included in Appendix D.

together (Figure 2(c)), but GCN-LPA still distinguishes the two clusters (Figure 2(d)), because it is better at “denoising” undesirable edges based on the supervised signal of labels.

It is worth noticing that the above analysis only works for homophilic graphs, i.e., two connected nodes are likely to have the same label. However, some real-world graphs do not satisfy the homophily property, (e.g., airline graphs where nodes are airports and edges are airlines),

Table 1. Statistics for All Datasets in Node Classification Task

	Cora	Citeseer	Pubmed	Coauthor-CS	Coauthor-Phy
# nodes	2,708	3,327	19,717	18,333	34,493
# edges	5,278	4,552	44,324	81,894	247,962
# classes	7	6	3	15	5
dimension of features	1,433	3,703	500	6,805	8,415
Intra-class edge rate	81.0%	73.6%	80.2%	80.8%	93.1%
Labeled node rate	5.2%	3.6%	0.3%	1.6%	0.3%

which makes our model inapplicable to these graphs. The exploration of applying GCN-LPA to non-homophilic graphs is left as future work.

6 EXPERIMENTS

We evaluate our model and present its performance in two tasks: semi-supervised node classification and knowledge-graph-aware recommendation. The code of GCN-LPA is available at https://github.com/hwwang55/GCN-LPA.

6.1 Semi-Supervised Node Classification

6.1.1 Datasets. We use the following five datasets in our experiments. Cora, Citeseer, and Pubmed [27] are citation networks, where nodes correspond to documents, edges correspond to citation links, and each node has a sparse bag-of-words feature vector as well as a class label. We also use two co-authorship networks [28], Coauthor-CS and Coauthor-Phy, where nodes are authors and an edge indicates that two authors co-authored a paper. Node features represent paper keywords for each author's papers, and class labels indicate most active fields of study for each author.

Statistics of the five datasets are shown in Table 1. We also calculate the intra-class edge rate (the fraction of edges that connect two nodes within the same class), which is significantly higher than inter-class edge rate in all networks. The finding supports our claim in Section 5 that node classification benefits from intra-class edges in a graph.

6.1.2 Baselines. We compare against the following baselines in our experiments. Logistic Regression is feature-based methods that do not consider the graph structure. We set solver = 'lbfgs' for LR in the Python sklearn package. LPA [49], however, only consider the graph structure and ignore node features. We set the iteration of LPA as 20. We also compare with several GNNs: GCN [12], Graph Attention Network (GAT), Jumping Knowledge Network (JK-Net) [40], Graph Isomorphism Network (GIN) [39], and Graph Diffusion Convolution (GDC) [14] (with GCN as the base model). There hyper-parameter settings are set as default in their original open-source codes. In addition, we propose three variants that also combines GCN and LPA: GCN-LPA(T), which learns the optimal adjacency matrix A by minimizing L_{lpa} first, then freezes A and optimizes W by minimizing L_{gcn}; GCN-LPA(S), which simultaneously optimizes A and W as in Equation (15), but the gradient of A only propagates back from L_{lpa}; GCN+LPA, which simply adds predictions of GCN and LPA together.

6.1.3 Experimental Setup. Our experiments focus on the transductive setting where we only know labels of part of nodes but have access to the entire graph as well as features of all nodes.[3]

[3]Our method can be easily generalized to inductive setting if implemented using minibatch training like GraphSAGE [6]. To accommodate to this change, edge weights should be designed as a function of features of two endpoints, and the neighborhood should be sampled based on the (normalized) edge weights.

Table 2. Hyper-parameter Settings for All Datasets in Node Classification Task

	Cora	Citeseer	Pubmed	Coauthor-CS	Coauthor-Phy
Dimension of hidden layers	32	16	32	32	32
# GCN layers	5	2	2	2	2
# LPA iterations	5	5	1	2	3
LPA weight (λ)	10	1	1	2	1
L2 weight	1×10^{-4}	5×10^{-4}	2×10^{-4}	1×10^{-4}	1×10^{-4}
Dropout rate	0.2	0	0	0.2	0.2
Learning rate	0.05	0.2	0.1	0.1	0.05

We randomly sample 20 nodes per class as training set, 50 nodes per class as validation set, and the remaining nodes as test set. The weight of each edge is treated as a free variable during training. We train our model for 200 epochs using Adam [11] and report the test set accuracy when validation set accuracy is maximized. Each experiment is repeated 10 times, and we report the mean and the 95% confidence interval. We initialize weights according to Reference [4] and row-normalize input features. During training, we apply L2 regularization to the transformation matrices and use the dropout technique [30].

The detailed hyper-parameter settings of GCN-LPA on all datasets are listed in Table 2. In GCN-LPA, we set the dimension of all hidden layers as the same. Note that the number of GCN layers and the number of LPA iterations can actually be different, since GCN and LPA are implemented as two independent modules. We use grid search to determine hyper-parameters on Cora, and fine-tune the hyper-parameters on other datasets, i.e., varying one hyper-parameter per time to see if the performance can be further improved. The search spaces for all hyper-parameters are listed follows:

- Dimension of hidden layers: $\{8, 16, 32\}$;
- # GCN layers: $\{1, 2, 3, 4, 5, 6\}$;
- # LPA iterations: $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$;
- LPA weight (λ): $\{0, 1, 2, 5, 10, 15, 20\}$;
- L2 weight: $\{10^{-7}, 2 \times 10^{-7}, 5 \times 10^{-7}, 10^{-6}, 2 \times 10^{-6}, 5 \times 10^{-6}, 10^{-5}, 2 \times 10^{-5}, 5 \times 10^{-5}, 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}, 10^{-3}\}$;
- Dropout rate: $\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$;
- Learning rate: $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.5\}$.

6.1.4 Comparison with Baselines. The results of node classification are summarized in Table 3. Table 3 indicates that only using node features (Logistic Regression) or graph structure (LPA) will lead to information loss and cannot fully exploit datasets. The results demonstrate that our proposed GCN-LPA model surpasses state-of-the-art GNN baselines. We are able to improve upon the best baseline by 0.8%, 1.0%, 0.8, and 1.6% on Citeseer, Pubmed, Coauthor-CS, and Coauthor-Phy, respectively (all percentages are absolute gains). We notice that GDC is a strong baseline on Cora, but it does not perform consistently well on other datasets. In addition, regarding the three variants of our method: GCN+LPA does not perform well, since it utilizes the prediction of LPA directly, making its performance limited by LPA; GCN-LPA(T) also performs worse than GCN-LPA, which is probably due to the reason that learning the optimal adjacency matrix first will make the edge weights prone to overfitting the training node labels; GCN-LPA(S) achieves almost the same performance as GCN-LPA. But we observe that it takes more time for GCN-LPA(S) to converge compared with GCN-LPA, since the adjacency matrix is updated based only on the signal from L_{lpa} without L_{gcn}.

Table 3. Mean and the 95% Confidence Intervals of Test Set Accuracy for All Methods and Datasets in Node Classification Task

Method	Cora	Citeseer	Pubmed	Coauthor-CS	Coauthor-Phy
Logistic Regression	57.0 ± 1.6	61.2 ± 1.5	64.0 ± 2.3	86.1 ± 0.8	86.7 ± 1.3
LPA	74.4 ± 2.0	67.5 ± 1.3	70.6 ± 3.4	73.8 ± 1.4	86.2 ± 1.5
GCN	81.4 ± 0.8	<u>71.8</u> ± 1.3	<u>77.6</u> ± 2.0	90.9 ± 0.5	92.3 ± 0.9
GAT	80.8 ± 1.2	71.3 ± 1.4	76.8 ± 1.6	90.4 ± 0.5	92.2 ± 0.7
JK-Net	81.1 ± 1.2	70.3 ± 1.0	<u>77.4</u> ± 0.5	90.3 ± 0.3	90.8 ± 0.6
GIN	74.2 ± 0.9	60.5 ± 1.1	73.3 ± 1.0	84.2 ± 1.1	87.1 ± 0.9
GDC	**83.2** ± 0.8	<u>72.0</u> ± 0.9	<u>77.8</u> ± 0.7	91.1 ± 0.6	92.1 ± 0.4
GCN-LPA(T)	<u>82.6</u> ± 0.7	<u>72.4</u> ± 0.7	<u>78.2</u> ± 1.0	<u>91.5</u> ± 0.5	<u>93.1</u> ± 0.7
GCN-LPA(S)	<u>82.9</u> ± 0.8	<u>72.4</u> ± 0.9	<u>78.4</u> ± 0.9	**91.8** ± 0.4	<u>93.4</u> ± 0.8
GCN+LPA	78.3 ± 0.5	69.9 ± 1.1	74.0 ± 0.6	84.4 ± 0.8	89.9 ± 0.7
GCN-LPA	<u>83.1</u> ± 0.7	**72.6** ± 0.8	**78.6** ± 1.3	**91.8** ± 0.4	**93.6** ± 1.0

The best result is highlighted in bold, while the result falling within the confidence interval of the highest one is highlighted with underline.

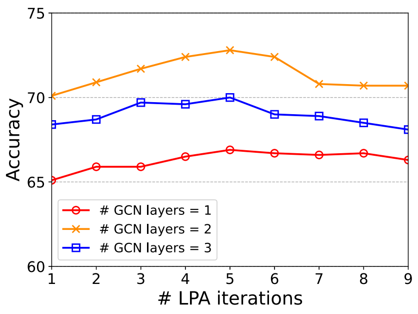


Fig. 3. Sensitivity to the number of LPA iterations on Citeseer dataset.

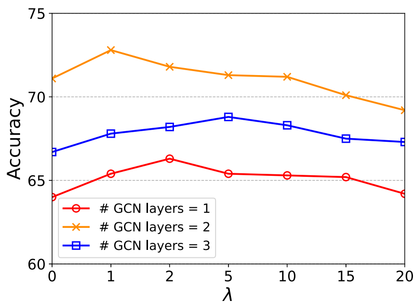


Fig. 4. Sensitivity to λ (weight of LPA loss) on Citeseer dataset.

6.1.5 *Efficacy of LPA Regularization.* We investigate the influence of the number of LPA iterations and the training weight of LPA loss term λ on the performance of classification. The results on Citeseer dataset are plotted in Figures 3 and 4, respectively, where each line corresponds to a given number of GCN layers in GCN-LPA. From Figure 3 we observe that the performance is boosted at first when the number of LPA iterations increases, then the accuracy stops increasing and decreases, since a large number of LPA iterations will include more noisy nodes. Figure 4 shows that training without the LPA loss term (i.e., $\lambda = 0$) is more difficult than the case where $\lambda = 1$–5, which justifies our aforementioned claim that it is hard for the GCN part to learn both transformation matrices W and edge weights A simultaneously without the assistance of LPA regularization.

6.1.6 *Influence of Labeled Node Rate.* To study the influence of labeled node rate on the performance of our model, we vary the ratio of labeled node rate on Citeseer from 2% to 60% while keeping the validation and test set fixed, and report the result in Table 4 (note that we do not deliberately keep the labels balanced for each label category to investigate the model performance in (possible) label-unbalanced settings). From Table 4 we observe that GCN-LPA outperforms GCN and LPA consistently, and the improvement achieved by GCN-LPA increases when labeled node rate is larger (from 0.6% to 2.1% compared with GCN). This is because GCN-LPA requires

Table 4. Accuracy of LPA, GCN, and GCN-LPA on Citeseer Dataset with Different Labeled Node Rate

Labeled node rate	2%	5%	10%	20%	40%	60%
LPA	65.0 ± 1.1	67.7 ± 1.5	68.2 ± 1.0	70.5 ± 1.4	71.6 ± 1.3	73.9 ± 0.8
GCN	69.6 ± 0.9	72.1 ± 1.2	72.4 ± 1.0	74.4 ± 0.6	75.7 ± 0.5	79.1 ± 0.8
GCN-LPA	**70.2** ± 0.7	**72.7** ± 1.0	**73.3** ± 0.8	**75.3** ± 1.1	**77.3** ± 0.8	**80.9** ± 1.0

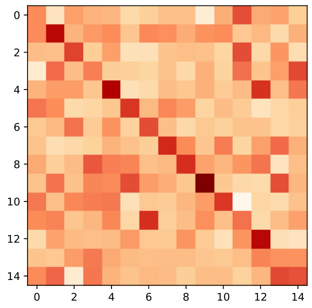


Fig. 5. Visualization of learned edge weights on Coauthor-CS dataset. The numbers along axes denote different node labels.

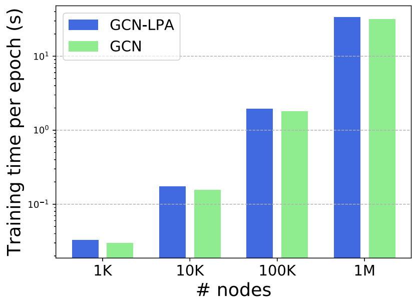


Fig. 6. Training time per epoch of GCN-LPA and GCN on random graphs.

node labels to calculate edge weights. Therefore, a larger labeled node rate will provide more information for identifying noisy edges.

6.1.7 *Visualization of Learned Edge Weights.* To intuitively understand what our model learns about edge weights, we split nodes in Coauthor-CS dataset into 15 groups according to their labels, and calculate the average weights of edges connecting every pair of node groups as well as the average weights of edges within every group. The results are shown in Figure 5, where darker color indicates higher average weights of edges. It is clear that values along the diagonal (intra-class edges weights) are significantly larger than off-diagonal values (inter-class edge weights) in general, which demonstrates that GCN-LPA is able to identify the importance of edges and distinguish inter-class and intra-class edges. The visualization results are similar for other datasets.

6.1.8 *Time Complexity.* We study the training time of GCN-LPA on random graphs. We use the one-hot identity vector as feature and $[0, 0]$ as label for each node. The size of training set and validation set is 100 and 200, respectively, while the rest is test set. The average number of neighbors for each node is 5, and the number of nodes is varied from one thousand to one million. We run GCN-LPA and GCN for 100 epochs on a Microsoft Azure virtual machine with 1 NVIDIA Tesla M60 GPU, 12 Intel Xeon CPUs (E5-2690 v3 @2.60 GHz), and 128 GB of RAM, using the same hyper-parameter setting as in Cora. The training time per epoch of GCN-LPA and GCN is presented in Figure 6. Our result shows that GCN-LPA requires only 9.2% extra training time on average compared to GCN.

Here we also provide theoretical analysis on time complexity. We first analyze the time complexity of GCN. Suppose that the graph has N nodes and E edges. For a given GCN layer, the input embedding dimension is D_i and the output embedding dimension is D_o. Therefore, the size of the three matrices $\tilde{A}$, $X^{(k)}$, $W^{(k)}$ in Equation (3) is $N \times N$, $N \times D_i$, and $D_i \times D_o$, respectively. The complexity of multiplying them together and applying a nonlinear function is $N^2 D_i + N D_i D_o$ and $N D_o$, respectively. The total complexity of one GCN layer is therefore $N^2 D_i + N D_i D_o + N D_o$. For

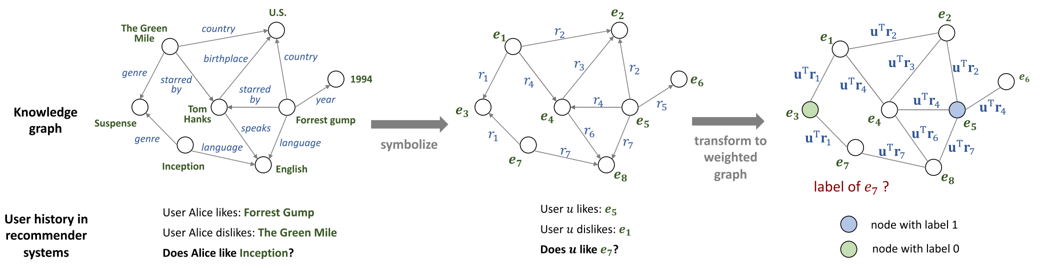


Fig. 7. Left: An instance of movie knowledge graph and the rating history of user Alice in recommender systems. Middle: The symbolized knowledge graph and user history. Right: Transforming the knowledge graph into a weighted graph, in which the weight of edge with relation r is $\mathbf{u}^\top\mathbf{r}$, where $\mathbf{u}$ and $\mathbf{r}$ are the trainable embeddings for user u and relation r, respectively. Nodes that user u likes are assigned with label 1 and nodes that user u dislikes are assigned with label 0. In this way, the KG-aware recommendation problem is transformed to a binary node classification problem.

GCN-LPA, the time complexity of one GCN-LPA layer contains an addition term for LPA, which is $N^2C + MC$ according to Equations (1) and (2), where C and M are the dimension of labels and the number of labeled nodes, respectively. Therefore, the ratio of additional time cost of GCN-LPA compared with GCN is $(N^2C + MC)/(N^2D_i + ND_iD_o + ND_o)$. Considering that the adjacency matrix is usually implemented as sparse matrix in practice, we replace N^2 with E in the above equation, which is now $(EC + MC)/(ED_i + ND_iD_o + ND_o) = (\frac{E}{N}C + \frac{M}{N}C)/(\frac{E}{N}D_i + D_iD_o + D_o)$ (by eliminating N) $= (\frac{d}{2}C + \frac{M}{N}C)/(\frac{d}{2}D_i + D_iD_o + D_o)$ (d is the average node degree) $\approx \frac{d}{2}C/(\frac{d}{2}D_i + D_iD_o + D_o)$ (because the labeled node rate $M/N \ll 1$). Note that d and C are usually much smaller than D_i and D_o, therefore, the additional time cost of LPA term is negligible in practice.

One possible limitation of this experiment is that, for fair comparison, we implement the GCN layers and the LPA term by ourselves using Tensorflow, but note that there may exist other GNN implementation that is specially optimized (e.g., Pytorch Geometric) and runs much faster than our implementation. This could be solved by applying the same optimization strategy to the implementation of the LPA regularization.

6.2 Knowledge-Graph-Aware Recommendation

Knowledge graphs (KGs) are a special type of graphs where nodes denote entities and edges denote relations among entities (see the left part of Figure 7 for an example). In many recommendation scenarios, items in recommender systems are also nodes in knowledge graphs. Therefore, knowledge graphs provide additional information about item-item relationship, which can be used to improve the performance of recommender systems. A typical KG-aware recommendation problem can be described as follows: Given a user set U and an item set I, we have some observed user-item interaction data $\{(u,i)\}_{u\in U, i\in I}$. In addition, suppose we also have a knowledge graph $G = \{(h,r,t)\}$ available, where h and t are two entities, and r is the relation between the two entities. Note that items are also nodes in KGs, i.e., $I \subset V$ where V is the set of KG nodes (entities). Our goal is to predict the potential subset of items that a particular user u may interact with.

The above problem can actually be seen as a binary node classification task: For a particular user, we only know labels of a part of nodes (items) in the KG, and the goal is to predict labels of remaining nodes (items) (see the right part of Figure 7 for an example). It is also worth noticing that our proposed GCN-LPA perfectly fits this problem, because a KG has no explicit edge weights and they need to be learned.

Table 5. Statistics for All Datasets in Recommendation Task

	MovieLens-20M	Book-Crossing	Last.FM	Dianping-Food
# users	138,159	19,676	1,872	2,298,698
# items	16,954	20,003	3,846	1,362
# interactions	13,501,622	172,576	42,346	23,416,418
# entities	102,569	25,787	9,366	28,115
# relations	32	18	60	7
# KG triples	499,474	60,787	15,518	160,519

To apply GCN-LPA to KG-aware recommendation problem, we first use GCN to propagate node features across the KG. Then the node labels can be calculated as $\sigma(\mathbf{u}^\top \mathbf{i})$, where $\mathbf{u}$ is the embedding of user u and $\mathbf{i}$ is the final representation of node (item) i output by the GCN. We also use LPA to propagate node labels across the KG, which serves as a regularization term to help learn edge weights. Here the weight of an edge is designed as $\mathbf{u}^\top \mathbf{r}$, where $\mathbf{u}$ is the embedding of user u and $\mathbf{r}$ is the embedding of the relation of this edge. The reason of such design is due to the fact that different users may care about different aspects of movies (e.g., some users care more about genre of movies while other users care more about their leading actors), so the edge weight $\mathbf{u}^\top \mathbf{r}$ is able to characterize personal preference of users.

6.2.1 Datasets. We utilize the following four datasets in our experiments for movie, book, music, and restaurant recommendations, respectively. **MovieLens-20M** is a widely used benchmark dataset in movie recommendations, which consists of approximately twenty million explicit ratings (ranging from 1 to 5) on the MovieLens website. **Book-Crossing** contains one million ratings (ranging from 0 to 10) of books in the Book-Crossing community. **Last.FM** contains musician listening information from a set of two thousand users from Last.fm online music system. **Dianping-Food** contains over ten million interactions (including clicking, buying, and adding to favorites) between approximately two million users and one thousand restaurants.

The statistics for all datasets are shown in Table 5.

6.2.2 Baselines. We compare our model with the following baselines for recommender systems, in which the first two baselines are KG-free while the rest are KG-aware methods. **SVD** [17] is a classic CF-based model using inner product to model user-item interactions. We use the unbiased version (i.e., the predicted engaging probability is modeled as $y_{uv} = \mathbf{u}^\top \mathbf{v}$). The dimension and learning rate for the four datasets are set as: $d = 8, \eta = 0.5$ for MovieLens-20M, Book-Crossing; $d = 8, \eta = 0.1$ for Last.FM; $d = 32, \eta = 0.1$ for Dianping-Food. **LibFM** [24] is a widely used feature-based factorization model for **click-through rate (CTR)** prediction. We concatenate user ID and item ID as input for LibFM. The dimension is set as $\{1, 1, 8\}$ and the number of training epochs is 50 for all datasets. **LibFM + TransE** extends LibFM by attaching an entity representation learned by TransE [2] to each user-item pair. The dimension of TransE is 32 for all datasets. **PER** [42] is a representative of path-based methods, which treats the KG as heterogeneous information networks and extracts meta-path-based features to represent the connectivity between users and items. The settings of dimension and learning rate are the same as SVD. **CKE** [43] is a representative of embedding-based methods, which combines CF with structural, textual, and visual knowledge in a unified framework. We implement CKE as CF plus a structural knowledge module in this article. The dimension of embedding for the four datasets are 64, 128, 64, 64. The training weight for KG part is 0.1 for all datasets. The learning rate are the same as in SVD. **RippleNet** [34] is a representative of hybrid methods, which is a memory-network-like [37] approach that propagates users' preferences on the KG for recommendation. The hyper-parameter settings for RippleNet

Table 6. Hyper-parameter Settings for the Four Datasets in Recommendation Task

	MovieLens-20M	Book-Crossing	Last.FM	Dianping-Food
Hidden layer dim.	32	64	16	8
# GCN layers	1	2	1	2
# LPA iterations	1	2	1	2
LPA weight (λ)	1.0	0.5	0.1	0.5
L2 weight	10^{-7}	2×10^{-5}	10^{-4}	10^{-7}
Learning rate	2×10^{-2}	2×10^{-4}	5×10^{-4}	2×10^{-2}

are as follows: $d = 8$, $H = 2$, $\lambda_1 = 10^{-6}$, $\lambda_2 = 0.01$, $\eta = 0.01$ for MovieLens-20M; $d = 16$, $H = 3$, $\lambda_1 = 10^{-5}$, $\lambda_2 = 0.02$, $\eta = 0.005$ for Last.FM; $d = 32$, $H = 2$, $\lambda_1 = 10^{-7}$, $\lambda_2 = 0.02$, $\eta = 0.01$ for Dianping-Food. **KGIN** [36] models user intents as an attentive combination of knowledge graph relations, and encourages the independence of different intents for better model capability and interpretability. We set the learning rate to 10^{-7}, the embedding size to 64, the number of layers to 3, the number of user intents to 4, the coefficient of independence modeling to 10^{-4}, and the coefficient of L2 regularization to 10^{-5}.

6.2.3 Experimental Setup. Hyper-parameter settings for the four datasets are given in Table 6, which are determined by optimizing *Recall*@10 on a validation set. The search spaces for hyper-parameters are as follows:

- Dimension of hidden layers: $\{4, 8, 16, 32, 64, 128\}$;
- # GCN layers: $\{1, 2, 3, 4\}$;
- # LPA iterations: $\{1, 2, 3, 4\}$;
- LPA weight (λ): $\{0, 0.01, 0.1, 0.5, 1, 5\}$;
- L2 weight: $\{10^{-9}, 10^{-8}, 10^{-7}, 2 \times 10^{-7}, 5 \times 10^{-7}, 10^{-6}, 2 \times 10^{-6}, 5 \times 10^{-6}, 10^{-5}, 2 \times 10^{-5}, 5 \times 10^{-5}, 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}, 10^{-3}\}$;
- Learning rate: $\{10^{-5}, 2 \times 10^{-5}, 5 \times 10^{-5}, 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}, 10^{-3}, 2 \times 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 2 \times 10^{-2}, 5 \times 10^{-2}, 10^{-1}\}$.

For each dataset, the ratio of training, validation, and test set is $6 : 2 : 2$. Each experiment is repeated 10 times, and the average performance is reported. All trainable parameters are optimized by Adam algorithm.

6.2.4 Comparison with Baselines. We evaluate our method in two experiment scenarios: (1) In top-K recommendation, we use the trained model to select K items with highest predicted click probability for each user in the test set, and choose *Recall@K* to evaluate the recommended sets. (2) In CTR prediction, we apply the trained model to predict each piece of user-item pair in the test set (including positive items and randomly selected negative items). We use *AUC* as the evaluation metric in CTR prediction.

The results of top-K recommendation and CTR prediction are presented in Tables 7 to 11, which show that GCN-LPA outperforms baselines by a significant margin. For example, the *AUC* of GCN-LPA surpasses the best baseline method by 1.3%, 1.7%, 1.5%, and 1.1% in MovieLens-20M, Book-Crossing, Last.FM, and Dianping-Food datasets, respectively (all percentages are absolute gains).

We also show daily performance of GCN-LPA and baselines on Dianping-Food to investigate performance stability. Figure 8 shows their *AUC* score from September 1, 2018 to September 30, 2018. We notice that the curve of GCN-LPA is consistently above baselines over the test period; Moreover, the performance of GCN-LPA is also with low variance, which suggests that GCN-LPA is also robust and stable in practice.

Table 7. The Results of *Recall@K* for MovieLens-20M Dataset in Recommendation Task

Model	R@2	R@10	R@50	R@100
SVD	3.6 ± 0.2	12.4 ± 0.2	27.7 ± 0.1	40.1 ± 0.2
LibFM	$\underline{3.8}$ ± 0.4	12.1 ± 0.5	27.0 ± 0.5	39.0 ± 0.6
LibFM + TransE	$\underline{4.0}$ ± 0.8	12.1 ± 1.0	28.1 ± 1.3	39.4 ± 1.1
PER	2.2 ± 0.7	7.6 ± 0.8	16.1 ± 0.7	24.5 ± 0.7
CKE	3.5 ± 0.5	10.7 ± 0.6	24.2 ± 0.6	32.3 ± 0.5
RippleNet	**4.5** ± 0.7	13.1 ± 0.9	27.5 ± 0.8	44.3 ± 0.6
KGIN	$\underline{4.1}$ ± 0.4	12.6 ± 0.4	27.1 ± 0.5	42.4 ± 0.3
GCN-LPA	$\underline{4.3}$ ± 0.5	**15.4** ± 0.5	**32.2** ± 0.3	**45.9** ± 0.4

Table 8. The Results of *Recall@K* for Book-Crossing Dataset in Recommendation Task

Model	R@2	R@10	R@50	R@100
SVD	2.5 ± 0.3	4.6 ± 0.5	7.8 ± 0.4	10.9 ± 0.4
LibFM	3.1 ± 0.5	6.1 ± 0.5	9.2 ± 0.5	12.5 ± 0.6
LibFM + TransE	3.7 ± 1.3	6.5 ± 1.2	9.7 ± 1.2	13.0 ± 1.3
PER	2.1 ± 0.8	4.0 ± 1.0	6.4 ± 1.1	7.0 ± 1.0
CKE	2.6 ± 0.4	5.2 ± 0.4	7.8 ± 0.5	11.2 ± 0.6
RippleNet	3.5 ± 0.6	7.4 ± 0.6	10.7 ± 0.7	12.8 ± 0.7
KGIN	3.3 ± 0.4	7.0 ± 0.5	10.2 ± 0.7	12.6 ± 0.5
GCN-LPA	**4.5** ± 0.4	**8.3** ± 0.5	**11.6** ± 0.4	**14.9** ± 0.4

Table 9. The Results of *Recall@K* for Last

Model	R@2	R@10	R@50	R@100
SVD	2.9 ± 0.5	9.8 ± 0.4	24.1 ± 0.6	33.2 ± 0.6
LibFM	3.2 ± 0.8	10.3 ± 0.8	26.0 ± 0.9	33.1 ± 0.7
LibFM + TransE	3.1 ± 1.5	10.2 ± 1.7	25.7 ± 1.8	32.6 ± 1.8
PER	1.4 ± 1.2	5.3 ± 1.0	11.6 ± 0.9	17.6 ± 1.0
CKE	2.2 ± 0.6	7.0 ± 0.5	18.1 ± 0.7	29.6 ± 0.7
RippleNet	3.1 ± 0.9	10.1 ± 0.8	24.0 ± 0.8	33.5 ± 0.9
KGIN	3.5 ± 0.4	10.6 ± 0.5	26.2 ± 0.7	33.8 ± 0.7
GCN-LPA	**4.4** ± 0.7	**12.1** ± 0.7	**27.6** ± 0.6	**37.0** ± 0.8

FM dataset in recommendation task.

6.2.5 Efficacy of LPA Regularization. To study the efficacy of LPA regularization, we fix the dimension of hidden layers as 4, 8, and 16, then vary λ from 0 to 5 to see how performance changes. The results of *Recall*@10 in Last.FM dataset are plotted in Figure 9. It is clear that the performance of GCN-LPA with a non-zero λ is better than $\lambda = 0$, which justifies our claim that LPA regularization can assist learning the edge weights in a KG. But note that a too-large λ is less favorable, since it overwhelms the overall loss and misleads the direction of gradients. According to the experiment results, we find that a λ between 0.1 and 1.0 is preferable in most cases.

6.2.6 Results in Cold-start Scenarios. To investigate the performance of GCN-LPA in cold-start scenarios, we vary the size of training set r of MovieLens-20M from $r = 100\%$ to $r = 20\%$ (while the validation and test set are kept fixed), and report the results of *AUC* in Table 12. When $r = 20\%$,

Table 10. The Results of *Recall@K* for Dianping-food Dataset in Recommendation Task

Model	R@2	R@10	R@50	R@100
SVD	3.9 ± 0.2	15.1 ± 0.3	32.8 ± 0.3	45.2 ± 0.3
LibFM	4.1 ± 0.5	15.6 ± 0.5	33.2 ± 0.6	44.9 ± 0.6
LibFM + TransE	4.3 ± 1.0	16.1 ± 0.8	**34.2** ± 0.6	45.4 ± 0.8
PER	2.5 ± 0.7	10.1 ± 0.7	25.7 ± 0.9	35.4 ± 0.6
CKE	3.2 ± 0.6	13.8 ± 0.7	30.4 ± 0.6	43.7 ± 0.5
RippleNet	4.1 ± 0.5	15.3 ± 0.6	32.8 ± 0.6	44.1 ± 0.7
KGIN	4.0 ± 0.5	15.5 ± 0.6	33.1 ± 0.6	44.5 ± 0.6
GCN-LPA	**4.8** ± 0.3	**17.2** ± 0.4	34.0 ± 0.3	**48.6** ± 0.3

Table 11. The Results of *AUC* for all Datasets in Recommendation Task

	MovieLens-20M	Book-Crossing	Last.FM	Dianping-Food
SVD	96.3 ± 0.4	67.3 ± 0.6	76.9 ± 0.3	83.9 ± 0.2
LibFM	95.5 ± 1.1	69.0 ± 0.6	77.8 ± 0.7	83.7 ± 0.5
LibFM + TransE	96.6 ± 0.8	69.8 ± 0.9	77.4 ± 0.6	83.9 ± 0.4
PER	83.1 ± 0.7	61.6 ± 0.5	63.3 ± 0.4	74.8 ± 0.4
CKE	92.4 ± 0.5	67.5 ± 0.7	74.5 ± 0.5	80.2 ± 0.6
RippleNet	96.1 ± 0.3	72.7 ± 0.4	77.0 ± 0.3	83.2 ± 0.2
KGIN	95.8 ± 0.3	72.0 ± 0.5	78.8 ± 0.4	83.5 ± 0.5
GCN-LPA	**97.9** ± 0.1	**74.4** ± 0.4	**80.3** ± 0.2	**85.1** ± 0.2

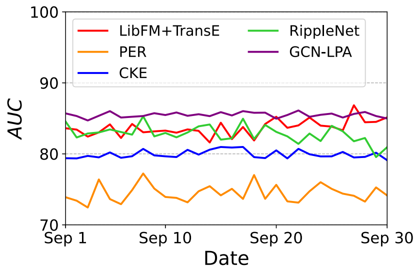


Fig. 8. Daily *AUC* of all KG-aware methods on Dianping-Food dataset in September 2018.

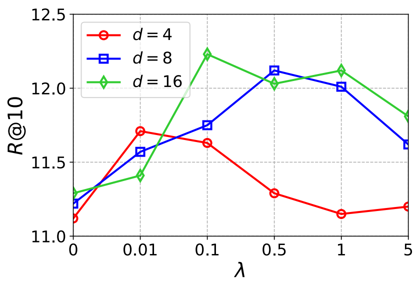


Fig. 9. Efficacy of LPA regularization on Last.FM dataset (d is the dimension of hidden layers).

AUC decreases by 8.1%, 5.3%, 5.2%, 2.9%, 2.6%, 4.0%, and 4.4% for the seven baselines compared to the model trained on full training data ($r = 100\%$), but the performance decrease of GCN-LPA is only 1.8%. This demonstrates that GCN-LPA still maintains predictive performance even when user-item interactions are sparse.

6.2.7 Hyper-parameters Sensitivity. We first analyze the sensitivity of GCN-LPA to the number of GNN layers and LPA iterations (these two numbers are set as the same in recommendation task). We vary this number from 1 to 4 while keeping other hyper-parameters fixed. The results are shown in Table 13. We find that the model performs poorly when the number is too large, which is because a large number of GCN layers or LPA iterations will mix too many entity embeddings in a given entity, which over-smoothes the representation learning on KGs.

Table 12. *AUC* of all Methods w.r.t. the Ratio of Training Set on MovieLens-20M Dataset

Ratio of training set r	20%	40%	60%	80%	100%
SVD	88.2 ± 0.5	91.3 ± 0.5	93.8 ± 0.5	95.5 ± 0.4	96.3 ± 0.4
LibFM	90.2 ± 1.2	92.3 ± 1.2	93.8 ± 1.0	95.0 ± 1.1	95.5 ± 1.1
LibFM+TransE	91.4 ± 0.9	93.5 ± 1.0	94.9 ± 0.9	96.0 ± 0.9	96.6 ± 0.8
PER	80.2 ± 0.6	81.4 ± 0.6	82.1 ± 0.8	82.8 ± 0.7	83.1 ± 0.7
CKE	89.8 ± 0.6	91.0 ± 0.7	91.6 ± 0.5	92.1 ± 0.6	92.4 ± 0.5
RippleNet	92.1 ± 0.4	93.7 ± 0.3	94.7 ± 0.4	95.5 ± 0.4	96.1 ± 0.3
KGIN	91.4 ± 0.3	93.2 ± 0.4	94.4 ± 0.4	95.1 ± 0.5	95.8 ± 0.3
GCN-LPA	**96.1** ± 0.1	**97.0** ± 0.2	**97.4** ± 0.1	**97.7** ± 0.0	**97.9** ± 0.1

Table 13. *Recall*@10 w.r.t. the Number of GCN Layers

# GCN layers	1	2	3	4
MovieLens-20M	**15.4** ± 0.5	14.6 ± 0.8	12.2 ± 1.3	1.1 ± 0.0
Book-Crossing	7.7 ± 0.4	**8.3** ± 0.5	4.3 ± 0.9	0.8 ± 0.2
Last.FM	**12.1** ± 0.7	10.6 ± 0.6	10.5 ± 0.6	5.7 ± 0.9
Dianping-Food	16.5 ± 0.3	**17.2** ± 0.4	6.1 ± 0.8	3.6 ± 0.9

Table 14. *Recall*@10 w.r.t. the Dimension of Hidden Layers

Hidden layer dim.	4	8	16	32	64	128
MovieLens-20M	13.4 ± 0.6	14.1 ± 0.6	14.3 ± 0.5	**15.4** ± 0.5	15.3 ± 0.4	15.1 ± 0.5
Book-Crossing	6.5 ± 0.7	7.3 ± 0.8	7.7 ± 0.7	8.1 ± 0.5	**8.3** ± 0.5	8.0 ± 0.6
Last.FM	11.1 ± 0.7	11.6 ± 0.9	**12.1** ± 0.7	10.9 ± 1.0	10.2 ± 0.8	10.7 ± 0.8
Dianping-Food	15.5 ± 0.6	**17.2** ± 0.4	16.7 ± 0.5	16.6 ± 0.5	16.3 ± 0.6	16.1 ± 0.4

We also examine the impact of the dimension of hidden layers on the performance of GCN-LPA. The result in shown in Table 14. We observe that the performance is boosted with the increase of the dimension at the beginning, because more bits in hidden layers can improve the model capacity. However, the performance drops when the number further increases, since a too-large dimension may overfit datasets.

6.2.8 Time Complexity. We also investigate the running time of our method with respect to the size of KG. We run experiments on a Microsoft Azure virtual machine with 1 NVIDIA Tesla M60 GPU, 12 Intel Xeon CPUs (E5-2690 v3 @2.60GHz), and 128 GB of RAM. The size of the KG is increased by up to five times the original one by extracting more triples from the original KG, and the running times of all methods on MovieLens-20M dataset are reported in Figure 10. Note that the trend of a curve matters more than the real values, since the values are largely dependent on the minibatch size and the number of epochs (yet we did try to align the configurations of all methods). The result show that GCN-LPA exhibits strong scalability even when the KG is large.

7 RELATED WORK

Edge weights play a key role in graph-based machine learning algorithms. In this section, we discuss three lines of related work that learn edge weights adaptively: locally linear embedding, label propagation algorithm, attention, and diffusion on graphs.

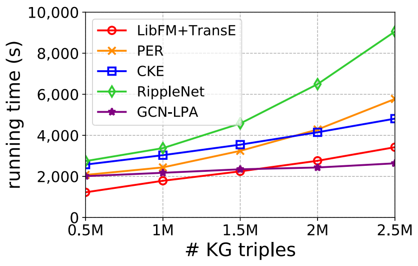


Fig. 10. Running time of all methods w.r.t. KG size on MovieLens-20M dataset.

7.1 Locally Linear Embedding

Locally linear embedding (LLE) [26] and its variants [16, 46] learn edge weights by constructing a linear dependency between a node and its neighbors and then use the learned edge weights to embed high-dimensional nodes into a low-dimensional space. Our work is similar to LLE in the aspect of transferring the knowledge of edge importance from one space to another, but the difference is that LLE is an unsupervised dimension reduction method that learns the graph structure based on local proximity only, while our work is semi-supervised and explores high-order relationship among nodes.

7.2 Label Propagation Algorithm

Classical LPA [47, 49] can only make use of node labels rather than node features. In contrast, adaptive LPA considers node features by making edge weights learnable. Typical techniques of learning edge weights include adopting kernel functions [21, 48] (e.g., $a_{ij} = \exp(-\sum_d (x_{id} - x_{jd})^2/\sigma_d^2)$ where d is dimensionality of features), minimizing neighborhood reconstruction error [10, 33], using leave-one-out loss [45], or imposing sparseness on edge weights [7]. However, in these LPA variants, node features are only used to assist learning the graph structure rather than explicitly mapped to node labels, which limits their capability in node classification. Another notable difference is that adaptive LPA learns edge weights by introducing the regularizations above, while our work takes LPA itself as regularization to learn edge weights.

7.3 Attention and Diffusion on Graphs

Our method is also conceptually connected to attention mechanism on graphs, in which an attention weight α_{ij} is learned between node v_i and v_j. For example, $\alpha_{ij} = \text{LeakyReLU}(\boldsymbol{a}^\top [W\mathbf{x}_i || W\mathbf{x}_j])$ in GAT [32], $\alpha_{ij} = a \cdot \cos(W\mathbf{x}_i, W\mathbf{x}_j)$ in AGNN [31], $\alpha_{ij} = (W_1\mathbf{x}_i)^\top W_2\mathbf{x}_j$ in GaAN [44], and $\alpha_{ij} = \boldsymbol{a}^\top \tanh(W_1\mathbf{x}_i + W_2\mathbf{x}_j)$ in GeniePath [22], where a and W are trainable variables. Our method is also similar to diffusion-based methods [1, 9, 13, 14, 38, 41]. Graph diffusion uses extended neighborhoods for aggregation in GNNs, which can be seen as learning a new adjacency matrix for a given graph. A significant difference between attention/diffusion mechanisms and our work is that attention/diffusion is learned based on feature similarity/graph topology, while we propose that edge weights should be consistent with the distribution of labels on the graph, which requires less handcrafting of the attention/diffusion function and is more task oriented.

7.4 Connection between LPA and GCN

Researchers also studied the connection between LPA and GCN from various perspectives. For example, Li et al. [19] studied the similarity of LPA and GCN in terms of low-pass filtering, and they proposed a a graph filtering framework that injects graph similarity into data features by taking them as signals on the graph and applying a low-pass graph filter to extract useful data representations for classification. Huang et al. [8] proposed the C&S method, which combines shallow models that ignore the graph structure with two simple post-processing steps that exploit correlation in the label structure: (1) an "error correlation" that spreads residual errors in training data to correct errors in test data and (2) a "prediction correlation" that smooths the predictions on the test data. They showed that their proposed method can exceed or match the performance of state-of-the-art GNNs. Dong et al. [3] studied the decoupled GCN (i.e., feature transformation and neighborhood aggregation are decoupled) and proved that the decoupled GCN is essentially the same as the two-step label propagation: first, propagating the known labels along the graph to generate pseudo-labels for the unlabeled nodes and, second, training normal neural network classifiers on the augmented pseudo-labeled data. Shi et al. proposed UniMP [29], which adopts a Graph Transformer network taking feature embedding and label embedding as input information for propagation. The difference between these work and ours is that we use LPA as a regularization term to assist learning the edge weights for GCN, which combines LPA and GCN more tightly and is shown to be effective both theoretically and empirically.

8 CONCLUSION AND FUTURE WORK

We studies the theoretical relationship between two types of well-known graph-based algorithms, label propagation algorithm and graph convolutional neural networks, from the perspectives of feature/label influence. We then propose a unified model GCN-LPA, which learns transformation matrices and edge weights simultaneously in GCN with the assistance of LPA regularizer. We also analyze why our unified model performs better than traditional GCN for node classification. Experiments on semi-supervised node classification and knowledge-graph-aware recommendation tasks demonstrate that our model outperforms state-of-the-art baselines, and it is also highly time-efficient with respect to the size of a graph.

We point out two possible directions as future work. First, it is worth studying why learning adjacency matrix and the parameters of GCN together performs better than learing them separately. Second, applying our method to the case where labels are extremely sparse is also a promising direction.

APPENDICES

A PROOF OF THEOREM 1

Before proving Theorem 1, we first give two lemmas that demonstrate the exact form of feature influence and label influence defined in this article. The relationship between feature influence and label influence can then be deduced from their exact forms.

Lemma 1. *Assume that the nonlinear activation function in GCN is ReLU. Let $P_k^{a \to b}$ be a path $[v^{(k)}, v^{(k-1)}, \dots, v^{(0)}]$ of length k from node v_a to node v_b, where $v^{(k)} = v_a$, $v^{(0)} = v_b$, and $v^{(i-1)} \in N(v^{(i)})$ for $i = k, \dots, 1$. Then we have*

$$\tilde{I}_f(v_a, v_b; k) = \sum_{P_k^{a \to b}} \prod_{i=k}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}, \tag{21}$$

where $\tilde{a}_{v^{(i-1)}, v^{(i)}}$ is the normalized weight of edge $(v^{(i)}, v^{(i-1)})$.

Proof. See Reference [40] for the detailed proof. □

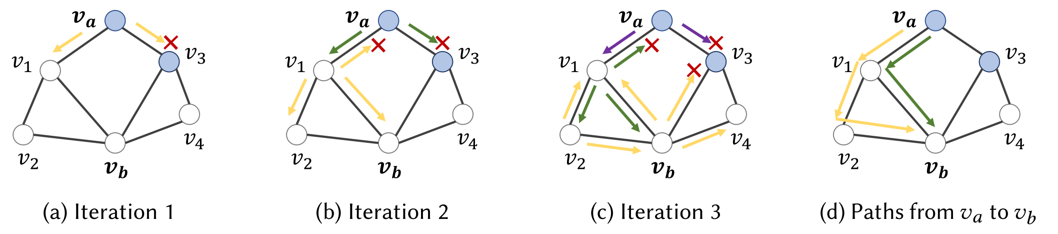


Fig. 11. An illustrating example of label propagation in LPA. Suppose labels are propagated for three iterations, and no self-loop exists. Blue nodes are labeled while white nodes are unlabeled. (a) v_a's label propagates to v_1 (yellow arrows). Note that the propagation of v_a's label to v_3 is cut off, since v_3 is labeled thus absorbing v_a's label. (b) v_a's label that propagated to v_1 further propagates to v_2 and v_b (yellow arrows). Meanwhile, v_a's label is reset to its initial value then propagates from v_a again (green arrows). (c) Label propagation in iteration 3. Purple arrows denote the propagation of v_a's label starting from v_a for the third time. (d) All possible paths of length no more than three from v_a to v_b containing unlabeled nodes only. Note that there is no path of length one from v_a to v_b.

The product term in Equation (21) is the probability of a given path $P_k^{a\to b}$. Therefore, the right-hand side in Equation (21) is the sum over probabilities of all possible paths of length k from v_a to v_b, which is the probability that a random walk starting at v_a ends at v_b after taking k steps.

Lemma 2. *Let $U_j^{a\to b}$ be a path $[v^{(j)}, v^{(j-1)}, \ldots, v^{(0)}]$ of length j from node v_a to node v_b, where $v^{(j)} = v_a$, $v^{(0)} = v_b$, $v^{(i-1)} \in N(v^{(i)})$ for $i = j, \ldots, 1$, and all nodes along the path are unlabeled except $v^{(0)}$. Then we have*

$$I_l(v_a, v_b; k) = \sum_{j=1}^{k} \sum_{U_j^{a\to b}} \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}, \tag{22}$$

where $\tilde{a}_{v^{(i-1)}, v^{(i)}}$ is the normalized weight of edge $(v^{(i)}, v^{(i-1)})$.

To intuitively understand this lemma, note that there are two differences between Lemma 1 and Lemma 2:

- In Lemma 1, $\tilde{I}_f(v_a, v_b; k)$ sums over all paths from v_a to v_b of length k, but in Lemma 2, $I_l(v_a, v_b; k)$ sums over all paths from v_a to v_b of length no more than k. The is because in LPA, v_b's label is reset to its initial value after each iteration, which means that the label of v_b serves as a constant signal that begins propagating in the graph again and again after each iteration.
- In Lemma 1 we consider all possible paths from v_a to v_b, but in Lemma 2, the paths are restricted to contain unlabeled nodes only. The reason here is the same as above: Since the labels of labeled nodes are reset to their initial values after each iteration in LPA, the influence of v_b's label will be absorbed in labeled nodes, and the propagation of v_b's label will be cut off at these nodes. Therefore, v_b's label can only flow to v_a along the paths with unlabeled nodes only. See Figure 11 for an illustrating example showing the label propagation in LPA.

Proof. As mentioned above, a significant difference between LPA and GCN is that all labeled nodes are reset to its original labels after each iteration in LPA. This implies that the initial label y_b of node v_b appears not only as $y_b^{(0)}$ but also as every $y_b^{(j)}$ for $j = 1, \ldots, k-1$. Therefore, the influence of y_b on $y_a^{(k)}$ is the cumulative influence of $y_b^{(j)}$ on $y_a^{(k)}$ for $j = 0, 1, \ldots, k-1$:

$$I_l(v_a, v_b; k) = \frac{\partial y_a^{(k)}}{\partial y_b} = \sum_{j=0}^{k-1} \frac{\partial y_a^{(k)}}{\partial y_b^{(j)}}. \tag{23}$$

According to the updating rule of LPA, we have

$$\frac{\partial y_a^{(k)}}{\partial y_b^{(j)}} = \frac{\partial \sum_{v_z \in N(v_a)} \tilde{a}_{az} y_z^{(k-1)}}{\partial y_b^{(j)}} = \sum_{v_z \in N(v_a)} \tilde{a}_{az} \frac{\partial y_z^{(k-1)}}{\partial y_b^{(j)}}. \tag{24}$$

In the above equation, the derivative $\frac{\partial y_a^{(k)}}{\partial y_b^{(j)}}$ is decomposed into the weighted average of $\frac{\partial y_z^{(k-1)}}{\partial y_b^{(j)}}$, where v_z traverses all neighbors of v_a. For those v_z's that are initially labeled, $y_z^{(k-1)}$ is reset to their initial labels in each iteration. Therefore, they are always constant and independent of $y_b^{(j)}$, meaning that their derivatives w.r.t. $y_b^{(j)}$ are zero. So we only need to consider the terms where v_z is an unlabeled node:

$$\frac{\partial y_a^{(k)}}{\partial y_b^{(j)}} = \sum_{v_z \in N(v_a), z>m} \tilde{a}_{az} \frac{\partial y_z^{(k-1)}}{\partial y_b^{(j)}}, \tag{25}$$

where $z > m$ means v_z is unlabeled. To intuitively understand Equation (25), one can imagine that we perform a random walk starting from node v_a for one step, where the "transition probability" is the edge weights $\tilde{a}$, and all nodes in this random walk are restricted to unlabeled nodes only. Note that we can further decompose every $y_z^{(k-1)}$ in Equation (25) in the way similar to what we do for $y_a^{(k)}$ in Equation (24). So the expansion in Equation (25) can be performed iteratively until the index k decreases to j. This is equivalent to performing all possible random walks for $k-j$ steps starting from v_a, where all nodes but the last in the random walk are restricted to be unlabeled nodes:

$$\frac{\partial y_a^{(k)}}{\partial y_b^{(j)}} = \sum_{v_z \in V} \sum_{U_{k-j}^{a \to z}} \left(\prod_{i=k-j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}} \right) \frac{\partial y_z^{(j)}}{\partial y_b^{(j)}}, \tag{26}$$

where v_z in the first summation term is the end node of a random walk, $U_{k-j}^{a \to z}$ in the second summation term is an unlabeled-nodes-only path from v_a to v_z of length $k-j$, and the product term is the probability of a given path $U_{k-j}^{a \to z}$. Consider the last term $\frac{\partial y_z^{(j)}}{\partial y_b^{(j)}}$ in Equation (26). We know that $\frac{\partial y_z^{(j)}}{\partial y_b^{(j)}} = 0$ for all $z \neq b$ and $\frac{\partial y_z^{(j)}}{\partial y_b^{(j)}} = 1$ for $z = b$, which means that only those random-walk paths that end exactly at v_b (i.e., the end node v_z is exactly v_b) count for the computation in Equation (26). Therefore, we have

$$\frac{\partial y_a^{(k)}}{\partial y_b^{(j)}} = \sum_{U_{k-j}^{a \to b}} \prod_{i=k-j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}, \tag{27}$$

where $U_{k-j}^{a \to b}$ is a path from v_a to v_b of length $k-j$ containing only unlabeled nodes except v_b. Substituting the right-hand term of Equation (23) with Equation (27), we obtain that

$$I_l(v_a, v_b; k) = \sum_{j=0}^{k-1} \sum_{U_{k-j}^{a \to b}} \prod_{i=k-j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}} = \sum_{j=1}^{k} \sum_{U_j^{a \to b}} \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}. \tag{28}$$

□

Now Theorem 1 can be proved by combining Lemma 1 and 2:

Proof. Suppose that whether a node is labeled or not is independent of each other for the given graph. Then we have

$$\begin{aligned}
&\mathbb{E}\Big[I_l(v_a, v_b; k)\Big] \\
&= \mathbb{E}\left[\sum_{j=1}^{k} \sum_{U_j^{a\to b}} \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}\right] \\
&= \sum_{j=1}^{k} \mathbb{E}\left[\sum_{U_j^{a\to b}} \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}\right] \\
&= \sum_{j=1}^{k} \sum_{P_j^{a\to b}} \Pr\left(P_j^{a\to b} \text{ is an unlabeled-nodes-only path}\right) \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}} \\
&= \sum_{j=1}^{k} \sum_{P_j^{a\to b}} \beta^j \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}} = \sum_{j=1}^{k} \beta^j \tilde{I}_f(v_a, v_b; j).
\end{aligned} \tag{29}$$

□

B PROOF OF THEOREM 2

Proof. Denote the set of labels as L. Since different label dimensions in $y_a^{(\cdot)}$ do not interact with each other when running LPA, the value of the y_ath dimension in $y_a^{(\cdot)}$ (denoted by $y_a^{(\cdot)}[y_a]$) comes only from the nodes with initial label y_a. It is clear that

$$y_a^{(k)}[y_a] = \sum_{v_b : y_b = y_a} \sum_{j=1}^{k} \sum_{U_j^{a\to b}} \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}, \tag{30}$$

which equals $\sum_{v_b : y_b = y_a} I_l(v_a, v_b; k)$ according to Lemma 2. Therefore, we have

$$\Pr(\hat{y}_a = y_a) = \frac{y_a^{(k)}[y_a]}{\sum_{i \in L} y_a^{(k)}[i]} \propto y_a^{(k)}[y_a] = \sum_{v_b : y_b = y_a} I_l(v_a, v_b; k) \tag{31}$$

□

C PROOF OF THEOREM 3

In this proof we assume that the dimension of node representations is one, but note that the conclusion can be easily generalized to the case of multi-dimensional representations, since the function $D(\mathbf{x})$ can be decomposed into the sum of one-dimensional cases. In the following of this proof, we still use bold notations $\mathbf{x}_i^{(k)}$ and $\mathbf{h}_i^{(k)}$ to denote node representations, but keep in mind that they are scalars rather than vectors.

We give two lemmas before proving Theorem 3. The first one is about the gradient of $D(\mathbf{x})$:

Lemma 3. $\mathbf{h}_i^{(k)} = \mathbf{x}_i^{(k)} - \frac{\partial D(\mathbf{x}^{(k)})}{\partial \mathbf{x}_i^{(k)}}$.

Proof. $\mathbf{x}_i^{(k)} - \frac{\partial D(\mathbf{x}^{(k)})}{\partial \mathbf{x}_i^{(k)}} = \mathbf{x}_i^{(k)} - \sum_{v_j \in N(v_i)} \tilde{a}_{ij} \left(\mathbf{x}_i^{(k)} - \mathbf{x}_j^{(k)}\right) = \sum_{v_j \in N(v_i)} \tilde{a}_{ij} \mathbf{x}_j^{(k)} = \mathbf{h}_i^{(k)}$. □

It is interesting to see from Lemma 3 that the aggregation step in GCN is equivalent to running gradient descent for one step with a step size of one. However, this is not able to guarantee that $D(\mathbf{h}^{(k)}) \leq D(\mathbf{x}^{(k)})$, because the step size may be too large to reduce the value of D.

The second lemma is about the Hessian of $D(\mathbf{x})$:

Lemma 4. *$\nabla^2 D(\mathbf{x}) \preceq 2I$, or equivalently, $2I - \nabla^2 D(\mathbf{x})$ is a positive semidefinite matrix.*

Proof. We first calculate the Hessian of $D(\mathbf{x}) = \frac{1}{2}\sum_{v_i, v_j} \tilde{a}_{ij} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$:

$$\nabla^2 D(\mathbf{x}) = \begin{bmatrix} 1-\tilde{a}_{11} & -\tilde{a}_{12} & \cdots & -\tilde{a}_{1n} \\ -\tilde{a}_{21} & 1-\tilde{a}_{22} & \cdots & -\tilde{a}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\tilde{a}_{n1} & -\tilde{a}_{n2} & \cdots & 1-\tilde{a}_{nn} \end{bmatrix} = I - D^{-1}A. \tag{32}$$

Therefore, $2I - \nabla^2 D(\mathbf{x}) = I + D^{-1}A$. Since $D^{-1}A$ is Markov matrix (i.e., each entry is non-negative and the sum of each row is one), its eigenvalues are within the range $[-1, 1]$, so the eigenvalues of $I + D^{-1}A$ are within the range $[0, 2]$. Therefore, $I + D^{-1}A$ is a positive semidefinite matrix, and we have $\nabla^2 D(\mathbf{x}) \preceq 2I$. □

We can now prove Theorem 3:

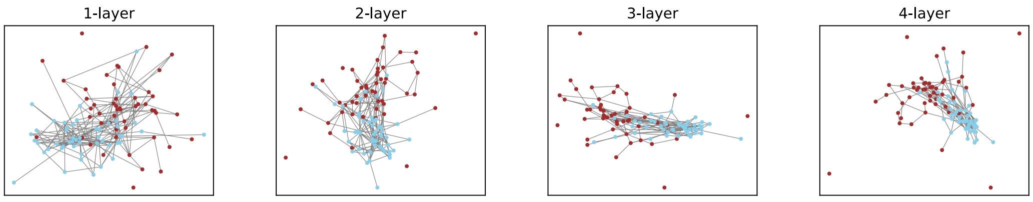
Proof. Since D is a quadratic function, we perform a second-order Taylor expansion of D around $\mathbf{x}^{(k)}$ and obtain the following inequality:

$$\begin{aligned} D\left(\mathbf{h}^{(k)}\right) &= D\left(\mathbf{x}^{(k)}\right) + \nabla D\left(\mathbf{x}^{(k)}\right)^\top \left(\mathbf{h}^{(k)} - \mathbf{x}^{(k)}\right) + \frac{1}{2}\left(\mathbf{h}^{(k)} - \mathbf{x}^{(k)}\right)^\top \nabla^2 D(\mathbf{x}) \left(\mathbf{h}^{(k)} - \mathbf{x}^{(k)}\right) \\ &= D\left(\mathbf{x}^{(k)}\right) - \nabla D\left(\mathbf{x}^{(k)}\right)^\top \nabla D\left(\mathbf{x}^{(k)}\right) + \frac{1}{2}\nabla D\left(\mathbf{x}^{(k)}\right)^\top \nabla^2 D(\mathbf{x}) \nabla D\left(\mathbf{x}^{(k)}\right) \\ &\leq D\left(\mathbf{x}^{(k)}\right) - \nabla D\left(\mathbf{x}^{(k)}\right)^\top \nabla D\left(\mathbf{x}^{(k)}\right) + \nabla D\left(\mathbf{x}^{(k)}\right)^\top \nabla D\left(\mathbf{x}^{(k)}\right) \\ &= D\left(\mathbf{x}^{(k)}\right). \end{aligned} \tag{33}$$

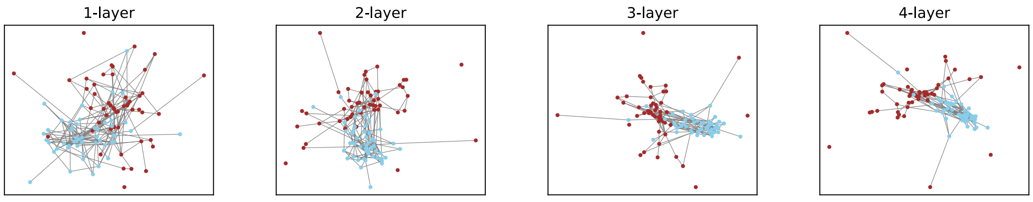
□

D MORE VISUALIZATION RESULTS ON A SUBGRAPH OF CORA

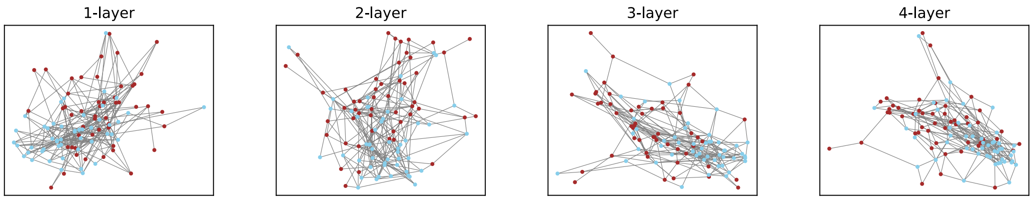
Figure 12 illustrates more visualization of GCN and GCN-LPA on a subgraph of Cora. In each subfigure, we vary the number of layers from 1 to 4 to examine how the learned representations evolve. The dimension of hidden layers and output layer is 2. The transformation matrices are uniformly initialized within range $[-1, 1]$. We use sigmoid function as the nonlinear activation function. Comparing the four figures in each row, we conclude that the aggregation step and transformation step in GCN and GCN-LPA do benefit the separation of different classes. Comparing Figures 12(a) and 12(c) (or Figures 12(b) and 12(d)), we conclude that more inter-class edges will make the separation harder for GCN (or GCN-LPA). Comparing Figures 12(a) and 12(b) (or Figures 12(c) and 12(d)), we conclude that GCN-LPA is more noise-resistant than GCN, therefore, GCN-LPA can better differentiate classes and identify clustering substructures.



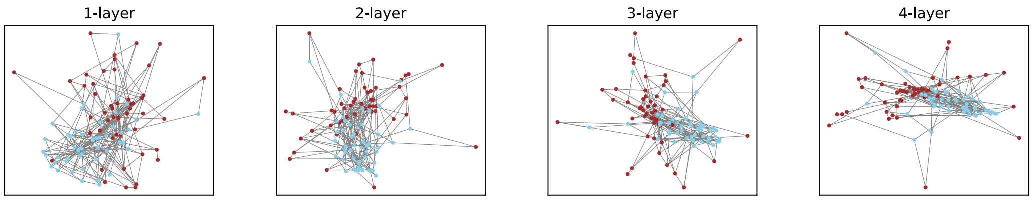
(a) GCN on the original subgraph



(b) GCN-LPA on the original subgraph



(c) GCN on the noisy subgraph



(d) GCN-LPA on the noisy subgraph

Fig. 12. Visualization of GCN and GCN-LPA with $1 \sim 4$ layers on a subgraph of Cora.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *Proceedings of the 36th International Conference on Machine Learning*. 21–29.

[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*. 2787–2795.

[3] Hande Dong, Jiawei Chen, Fuli Feng, Xiangnan He, Shuxian Bi, Zhaolin Ding, and Peng Cui. 2021. On the equivalence of decoupled graph convolution network and label propagation. In *Proceedings of the Web Conference 2021*. 3651–3662.

[4] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics.*

[5] Chen Gong, Dacheng Tao, Wei Liu, Liu Liu, and Jie Yang. 2016. Label propagation via teaching-to-learn and learning-to-teach. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 6 (2016), 1452–1465.

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

[7] Cheng Hong, Zicheng Liu, and Jie Yang. 2009. Sparsity induced similarity measure for label propagation. In *Proceedings of the 12th IEEE International Conference on Computer Vision*. IEEE.

[8] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. 2021. Combining label propagation and simple models out-performs graph neural networks. In *Proceedings of the 9th International Conference on Learning Representations*.

[9] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. 2019. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 11313–11320.

[10] Masayuki Karasuyama and Hiroshi Mamitsuka. 2013. Manifold-based similarity adaptation for label propagation. In *Advances in Neural Information Processing Systems*.

[11] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*.

[12] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*.

[13] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. In *Proceedings of the 7th International Conference on Learning Representations*.

[14] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems*. 13354–13366.

[15] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning*.

[16] Deguang Kong, Chris Ding, Heng Huang, and Feiping Nie. 2012. An iterative locally linear embedding algorithm. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*. Omnipress.

[17] Yehuda Koren. 2008. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 426–434.

[18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

[19] Qimai Li, Xiao-Ming Wu, Han Liu, Xiaotong Zhang, and Zhichao Guan. 2019. Label efficient semi-supervised learning via graph filtering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9582–9591.

[20] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S. Zemel. 2019. LanczosNet: Multi-scale deep graph convolutional networks. In *Proceedings of the 7th International Conference on Learning Representations*.

[21] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. 2019. Learning to propagate labels: Transductive propagation network for few-shot learning. In *Proceedings of the 7th International Conference on Learning Representations*.

[22] Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. 2019. GeniePath: Graph neural networks with adaptive receptive paths. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*.

[23] Meng Qu, Yoshua Bengio, and Jian Tang. 2019. GMNN: Graph markov neural networks. In *Proceedings of the 36th International Conference on Machine Learning*.

[24] Steffen Rendle. 2012. Factorization machines with libfm. *ACM Trans. Intell. Syst. Technol.* 3, 3 (2012), 57.

[25] Ryan A. Rossi, Luke K. McDowell, David William Aha, and Jennifer Neville. 2012. Transforming graph data for statistical relational learning. *J. Artif. Intell. Res.* 45 (2012), 363–441.

[26] Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290, 5500 (2000).

[27] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI Mag.* 29, 3 (2008).

[28] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. In *Neural Information Processing Systems Workshop on Relational Representation Learning*.

[29] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. 2020. Masked label prediction: Unified message passing model for semi-supervised classification. arXiv:2009.03509. Retrieved from https://arxiv.org/abs/2009.03509.

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014).

[31] Kiran K. Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. 2018. Attention-based graph neural network for semi-supervised learning. arXiv:1803.03735. Retrieved from https://arxiv.org/abs/1803.03735.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations*.

[33] Fei Wang and Changshui Zhang. 2008. Label propagation through linear neighborhoods. *IEEE Trans. Knowl. Data Eng.* 20, 1 (2008).

[34] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. RippleNet: Propagating user preferences on the knowledge graph for recommender systems. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 417–426.

[35] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 968–977.

[36] Xiang Wang, Tinglin Huang, Dingxian Wang, Yancheng Yuan, Zhenguang Liu, Xiangnan He, and Tat-Seng Chua. 2021. Learning intents behind interactions with knowledge graph for recommendation. In *Proceedings of the Web Conference 2021*. 878–887.

[37] Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks. In *Proceedings of the 3rd International Conference on Learning Representations*.

[38] Bingbing Xu, Huawei Shen, Qi Cao, Keting Cen, and Xueqi Cheng. 2019. Graph convolutional networks using heat kernel for semi-supervised learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 1928–1934.

[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations*.

[40] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning*.

[41] Liang Yang, Zesheng Kang, Xiaochun Cao, Di Jin, Bo Yang, and Yuanfang Guo. 2019. Topology optimization based graph convolutional network. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'19)*. 4054–4061.

[42] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. 2014. Personalized entity recommendation: A heterogeneous information network approach. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. ACM, 283–292.

[43] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 353–362.

[44] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. 2018. GaAN: Gated attention networks for learning on large and spatiotemporal graphs. arXiv:1803.07294. Retrieved from https://arxiv.org/abs/1803.07294.

[45] Xinhua Zhang and Wee S. Lee. 2007. Hyperparameter learning for graph based semi-supervised learning algorithms. In *Advances in Neural Information Processing Systems*.

[46] Zhenyue Zhang and Jing Wang. 2007. MLLE: Modified locally linear embedding using multiple weights. In *Advances in Neural Information Processing Systems*.

[47] Dengyong Zhou, Olivier Bousquet, Thomas N. Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *Advances in Neural Information Processing Systems*.

[48] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning*.

[49] Xiaojin Zhu, John Lafferty, and Ronald Rosenfeld. 2005. *Semi-supervised learning with graphs*. Ph.D. Dissertation. Carnegie Mellon University, School of Language Technologies Institute.

Received May 2021; revised September 2021; accepted September 2021