Unifying Graph Convolutional Neural Networks and Label Propagation

Anonymous authors
Paper under double-blind review

Abstract

Label Propagation (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 relation between LPA and GCN has not yet been investigated. Here we study the relationship between LPA and GCN in terms of two aspects: (1) *feature/label smoothing* where we analyze how the feature/label of one node is spread over its neighbors; And, (2) *feature/label influence* of how much the initial feature/label of one node influences the final feature/label of another node. Based on our theoretical analysis, we propose an end-to-end model that unifies GCN and LPA for node classification. 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 classification performance. Our model can also be seen as learning attention weights based on node labels, which is more task-oriented than existing feature-based attention models. In a number of experiments on real-world graphs, our model shows superiority over state-of-the-art GCN-based methods in terms of node classification accuracy.

1 Introduction

Consider the problem of node classification in a graph, where the goal is to learn a mapping $\mathcal{M} : \mathcal{V} \rightarrow \mathcal{L}$ from nodes $\mathcal{V}$ to labels $\mathcal{L}$. Solution to this problem is widely applicable to 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 and features vary *smoothly* over the edges of the graph. In particular, on the label side $\mathcal{L}$, node labels are propagated and aggregated along edges in the graph, which is known as *Label Propagation Algorithm* (LPA) (Zhu et al., 2005; Zhou et al., 2004; Zhang & Lee, 2007; Wang & Zhang, 2008; Karasuyama & Mamitsuka, 2013; Gong et al., 2017; Liu et al., 2019a); On the node side $\mathcal{V}$, node features are propagated along edges and transformed through neural network layers, which is known as *Graph Convolutional Neural Networks* (GCN) (Kipf & Welling, 2017; Hamilton et al., 2017; Li et al., 2018; Xu et al., 2018; Liao et al., 2019; Xu et al., 2019; Qu et al., 2019).

GCN and LPA are related in that they propagate features and labels on the two sides of the mapping $\mathcal{M}$, respectively. However, the relationship between GCN and LPA has not yet been investigated. Specifically, what is the theoretical relationship between GCN and LPA, and how can they be combined to develop a more accurate model for node classification in graphs?

Here we study the theoretical relationship between GCN and LPA from two viewpoints: (1) *Feature/label smoothing*, where we show that the intuition behind GCN/LPA is smoothing features/labels of nodes across the edges of the graph, i.e., one node's feature/label equals the weighted average of features/labels of its neighbors. We prove that if edge weights smooth the node features, they also smooth the node labels with guaranteed upper bound on the approximation error. And, (2) *feature/label influence*, where we quantify how much the initial feature/label of node v_b influences the output feature/label of node v_a in GCN/LPA by studying the Jacobian/gradient of node v_b with respect to node v_a, and then we also prove their quantitative relationship.

Based on the above 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 within the same class/label 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. 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 and do final classification. In GCN-LPA, we further combine the 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 classes. It is worth noticing that GCN-LPA can also be seen as learning *attention weights* for edges based on *node label information*, which requires less handcrafting and is more task-oriented than existing work that learns attention weights based on *node feature similarity* (Veličković et al., 2018; Thekumparampil et al., 2018; Zhang et al., 2018; Liu et al., 2019b). Experiments on five datasets indicate that our model outperforms state-of-the-art methods in terms of classification accuracy.

2 Unifying GCN and LPA

In this section, we first formulate the node classification problem and briefly introduce LPA and GCN. We then prove their relationship from the viewpoints of smoothing and influence. Based on our theoretical analysis, we then propose a unified model GCN-LPA, and we also analyze the reason why our model performs better than GCN in node classification.

2.1 Problem Formulation and Preliminaries

We begin by describing the problem of node classification on graphs and introducing notation. Consider a graph $\mathcal{G} = (\mathcal{V}, A, X, Y)$, where $\mathcal{V} = \{v_1, \cdots, v_n\}$ is the set of nodes, $A \in \mathbb{R}^{n \times n}$ is the adjacency matrix (self-loops are included), X is the feature matrix of nodes and Y is labels of nodes. a_{ij} (the ij-th entry of A) is the weight of the edge connecting v_i and v_j. $\mathcal{N}(v)$ denotes the set of immediate neighbors of node v in graph $\mathcal{G}$. Each node v_i has a feature vector $\mathbf{x}_i$ which is the i-th row of X, while only the first m nodes have labels $y_1, \cdots, y_m$ from a label set $\mathcal{L} = \{1, \cdots, c\}$. The goal is to learn a mapping $\mathcal{M} : \mathcal{V} \rightarrow \mathcal{L}$ and predict labels of unlabeled nodes.

Label Propagation Algorithm. LPA assumes that two connected nodes are likely to have the same label, and thus it propagates labels iteratively along the edges. Let $Y^{(k)} = [y_1^{(k)}, \cdots, y_n^{(k)}]^\top \in \mathbb{R}^{n \times c}$ be the soft label matrix in iteration $k > 0$, in which the i-th row $y_i^{(k)\top}$ denotes the predicted label distribution for node v_i in iteration k. When $k = 0$, the initial label matrix $Y^{(0)} = [y_1^{(0)}, \cdots, y_n^{(0)}]^\top$ consists of one-hot label indicator vectors $y_i^{(0)}$ for $i = 1, \cdots, m$ (i.e., labeled nodes) or zero vectors otherwise (i.e., unlabeled nodes). Let D be the diagonal degree matrix for A with entries $d_{ii} = \sum_j a_{ij}$. Then LPA (Zhu et al., 2005) in iteration k is formulated as the following two steps:

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

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

In Eq. (1), all nodes propagate their labels to their neighbors according to weights of edges. Then in Eq. (2), labels of all labeled nodes are reset to their initial values, because LPA wants to persist labels of nodes which are labeled so that unlabeled nodes do not overpower the labeled ones as the initial labels would otherwise fade away.

Graph Convolutional Neural Network. GCN is a multi-layer feedforward neural network that propagates and transforms node features across the graph. The layer-wise propagation rule of GCN is $X^{(k+1)} = \sigma(D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X^{(k)} W^{(k)})$, where $W^{(k)}$ is trainable weight matrix in the k-th layer, $\sigma(\cdot)$ is an activation function such as ReLU, and $X^{(k)} = [\mathbf{x}_1^{(k)}, \cdots, \mathbf{x}_n^{(k)}]^\top$ are the k-th layer node representations with $X^{(0)} = X$. To align with the above LPA, we use $D^{-1}A$ as the normalized adjacency matrix instead of the symmetric one $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ proposed by Kipf & Welling (2017).

Therefore, the feature propagation scheme of GCN in layer k is:

$$X^{(k+1)} = \sigma(D^{-1}AX^{(k)}W^{(k)}). \tag{3}$$

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

2.2 Feature Smoothing and Label Smoothing

The intuition behind both LPA and GCN is *smoothing* (Zhu et al., 2003; Li et al., 2018): In LPA, the final label of a node is the weighted average of labels of its neighbors: $y_i^{(\infty)} = \frac{1}{d_{ii}} \sum_{j \in \mathcal{N}(i)} a_{ij} y_j^{(\infty)}$; in GCN, the final node representation is also the weighted average of representations of its neighbors if we assume σ is identity function and $W^{(\cdot)}$ are identity matrices: $\mathbf{x}_i^{(\infty)} = \frac{1}{d_{ii}} \sum_{j \in \mathcal{N}(i)} a_{ij} \mathbf{x}_j^{(\infty)}$. Next we show the relationship between feature smoothing and label smoothing:

Theorem 1 (Relationship between feature smoothing and label smoothing) *Suppose that the latent ground-truth mapping $\mathcal{M} : \mathbf{x} \rightarrow y$ from node features to node labels is differentiable and satisfies L-Lipschitz constraint, i.e., $|\mathcal{M}(\mathbf{x}_1) - \mathcal{M}(\mathbf{x}_2)| \leq L\|\mathbf{x}_1 - \mathbf{x}_2\|_2$ for any $\mathbf{x}_1$ and $\mathbf{x}_2$ (L is a constant). If the edge weights $\{a_{ij}\}$ approximately smooth $\mathbf{x}_i$ over its immediate neighbors with error ϵ_i, i.e., $\mathbf{x}_i = \frac{1}{d_{ii}} \sum_{j \in \mathcal{N}(i)} a_{ij}\mathbf{x}_j + \epsilon_i$, then the edge weights $\{a_{ij}\}$ also approximately smooth y_i over its immediate neighbors with the following approximation error: $\big|y_i - \frac{1}{d_{ii}} \sum_{j \in \mathcal{N}(i)} a_{ij} y_j\big| \leq L\|\epsilon_i\|_2 + o\big(\max_{j \in \mathcal{N}(i)}(\|\mathbf{x}_j - \mathbf{x}_i\|_2)\big)$, where $o(\alpha)$ denotes a higher order infinitesimal than α.*

Proof of Theorem 1 is in Appendix A. Theorem 1 indicates that label smoothing is theoretically guaranteed by feature smoothing. Note that if we treat edge weights $\{a_{ij}\}$ learnable, then feature smoothing (i.e., $\epsilon_i \rightarrow 0$) can be directly achieved by keeping node features $\mathbf{x}_i$ fixed while setting $\{a_{ij}\}$ appropriately, without resorting to feature propagation in a multi-layer GCN. Therefore, a simple approach to exploit this theorem would be to learn $\{a_{ij}\}$ by reconstructing node feature $\mathbf{x}_i$ from its neighbors, then use the learned $\{a_{ij}\}$ to reconstruct node labels y_i (Karasuyama & Mamitsuka, 2013). This is equivalent to first minimizing the difference between $X^{(0)}$ and $X^{(1)}$ by learning $D^{-1}A$ in a one-layer GCN where σ and $W^{(0)}$ are identity, then plug the learned $D^{-1}A$ in LPA and run it for one iteration.

As shown in Theorem 1, the approximation error of labels is dominated by $L\|\epsilon_i\|_2$. However, this error could be fairly large in practice because: (1) The number of immediate neighbors for a given node may be too small to reconstruct its features perfectly, especially in the case where node features are high-dimensional and sparse. For example, in a citation network where node features are one-hot bag-of-words vectors, the feature of one article can never be precisely reconstructed if none of its neighboring articles contains the specific word that appears in this article. As a result, $\|\epsilon_i\|_2$ will be non-neglibible. This explains why it is beneficial to apply LPA and GCN for multiple iterations/layers in order to include information from farther away neighbors. (2) The ground-truth mapping $\mathcal{M}$ may not be sufficiently smooth due to the complex structure of latent manifold and possible noise, which fails to satisfy L-Lipschitz constraint. In other words, the constant L will be extremely large.

2.3 Feature Influence and Label Influence

To address the above concerns and extend our analysis, we next consider GCN and LPA with multiple layers/iterations, and do not impose any constraint on the ground-truth mapping $\mathcal{M}$.

Consider two nodes v_a and v_b in a graph. Inspired by Koh & Liang (2017) and Xu et al. (2018), 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 k-th 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 $\partial \mathbf{x}_a^{(k)} / \partial \mathbf{x}_b$: $I_f(v_a, v_b; k) = \left\| \mathbb{E}\left[\partial \mathbf{x}_a^{(k)} / \partial \mathbf{x}_b\right] \right\|_1$. The normalized feature influence is then defined as $\tilde{I}_f(v_a, v_b; k) = I_f(v_a, v_b; k) / \sum_{v_i \in \mathcal{V}} I_f(v_a, v_i; k)$.*

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 $[0, 1]$ (i.e., a binary classification) 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: $I_l(v_a, v_b; k) = \partial y_a^{(k)} / \partial y_b$.*

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

Theorem 2 (Relationship between feature influence and label influence) *Assume the activation function used in GCN is ReLU. 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:*

$$\mathbb{E}\left[I_l(v_a, v_b; k)\right] = \sum\nolimits_{j=1}^{k} \beta^j \tilde{I}_f(v_a, v_b; j). \tag{4}$$

Proof of Theorem 2 is in Appendix B. Intuitively, Theorem 2 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 2 provides the theoretical guideline for designing our unified model in the next subsection.

2.4 The Unified Model

Before introducing the proposed model, we first rethink the GCN method and see what an ideal node representation should be like. Since we aim to classify nodes, the perfect node representation would be such that nodes with the same label are embedded close 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 the GCN. We can therefore make edge strengths/weights trainable, then learn to increase the *intra-class feature influence* for each class i: $\sum_{v_a, v_b: y_a = i, y_b = i} \tilde{I}_f(v_a, v_b)$ 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 initial and output features, 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., $\sum_{v_a, v_b: y_a = i, y_b = i} I_l(v_a, v_b)$, according to Theorem 2. We further show that, by the following theorem, the intra-class label influence for a given node v_a is proportional to the probability that v_a is classified correctly by LPA:

Theorem 3 (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:*

$$\sum_{v_b : y_b = y_a} I_l(v_a, v_b; k) \propto \Pr\left(\hat{y}_a^{lpa} = y_a\right), \tag{5}$$

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

Proof of Theorem 3 is in Appendix C. Theorem 3 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 for node v_a. We can therefore first learn the optimal edge weights A^* by minimizing the loss of predicted labels by LPA:[1]

$$A^* = \arg\min_A L_{lpa}(A) = \arg\min_A \frac{1}{m} \sum_{v_a : a \leq m} J(\hat{y}_a^{lpa}, y_a), \tag{6}$$

where J is the cross-entropy loss, $\hat{y}_a^{lpa}$ and y_a are the predicted label distribution of v_a using LPA and the true one-hot label of v_a, respectively.[2] $a \leq m$ means v_a is labeled. The optimal A^* maximize the probability that each node is correctly labeled by LPA, thus also increasing the intra-class label influence (by Theorem 3) and intra-class feature influence (by Theorem 2). Then we can apply A^* and the corresponding D^* to a GCN to predict labels and learn optimal transformation matrices:

$$X^{(k+1)} = \sigma({D^*}^{-1} A^* X^{(k)} W^{(k)}), \quad k = 0, 1, \cdots, K-1, \tag{7}$$

$$W^* = \arg\min_W L_{gcn}(W, A^*) = \arg\min_W \frac{1}{m} \sum_{v_a : a \leq m} J(\hat{y}_a^{gcn}, y_a), \tag{8}$$

where $\hat{y}_a^{gcn}$ (which is the a-th row of $X^{(K)}$) is the predicted label distribution of v_a using the GCN specified in Eq. (7).

In practice, it is generally better to combine the above two steps together and train the whole model in an end-to-end fashion:

$$W^*, A^* = \arg\min_{W,A} \; L_{gcn}(W, A) + \lambda L_{lpa}(A), \tag{9}$$

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 the 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 the GCN. From this perspective, GCN-LPA also connects to Theorem 1 except that the knowledge transfer is in the other direction.

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 $M \circ A = A^*$. Each element M_{ij} can be set as either a free variable or a function of the nodes at edge endpoints, for example, $M_{ij} = \log\big(\exp(\mathbf{x}_i^\top \mathbf{H} \mathbf{x}_j) + 1\big)$ where $\mathbf{H}$ is a learnable kernel matrix for measuring feature similarity.

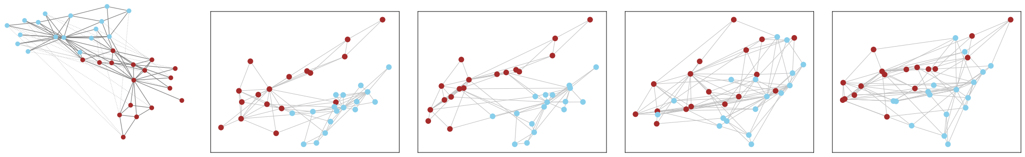
2.5 Analysis of GCN-LPA Model Behavior

In this subsection, 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: $\mathbf{x}_i^{(k+1)} = \sigma\big(\sum_{v_j \in \mathcal{N}(v_i)} \tilde{a}_{ij} \mathbf{x}_j^{(k)} W^{(k)}\big)$, 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: (1) In *aggregation* step, we calculate the aggregated representation $\mathbf{h}_i^{(k)}$ of all neighborhoods $\mathcal{N}(v_i)$: $\mathbf{h}_i^{(k)} = \sum_{v_j \in \mathcal{N}(v_i)} \tilde{a}_{ij} \mathbf{x}_j^{(k)}$; (2) In *transformation* step, the aggregated representation $\mathbf{h}_i^{(k)}$ is mapped to a new space by a transformation matrix and nonlinear function: $\mathbf{x}_i^{(k+1)} = \sigma\big(\mathbf{h}_i^{(k)} W^{(k)}\big)$. 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 4 (Shrinking property in GCN) *Let $D(\mathbf{x}) = \frac{1}{2} \sum_{v_i, v_j} \tilde{a}_{ij} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$ be a distance metric between the embeddings $\mathbf{x}$ of nodes. Then we have $D(\mathbf{h}^{(k)}) \leq D(\mathbf{x}^{(k)})$.*

[1] Here the optimal edge weights A^* share the same topology as the original graph $\mathcal{G}$, meaning that we do not add or remove edges from $\mathcal{G}$ but only learning the weights of existing edges. See the end of this subsection for more discussion.

[2] Here we somewhat abuse the notations for simplicity, since in Theorem 3 the two notations represent label category rather than label distribution. But the subtle difference can be easily distinguished based on context.



(a) Karate club network with noisy edges (b) GCN on the original network (c) GCN-LPA on the original network (d) GCN on the noisy network (e) GCN-LPA on the noisy network

Figure 1: Node embeddings of Zachary's karate club network trained on a node classification task (red vs. blue). Node coordinates in (b)-(e) are the embedding coordinates. Notice that GCN does not produce linearly separable embeddings ((b) vs. (c)), while GCN-LPA performs much better even in the presence of noisy edges ((d) vs. (e)). Additional visualizations are included in Appendix E.

Proof of Theorem 4 is in Appendix D. Theorem 4 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 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 labels reconstructed from multi-hop neighbors. This will force the model to increase weight/bandwidth of possible paths that connect nodes with the same label, so that labels can "flow" easily along these paths for the purpose of label reconstruction. In this way, GCN-LPA is able to identify potential intra-class edges and increase their weights to assist learning clustering structures.

To illustrate this, we apply a two-layer untrained GCN with randomly initialized transformation matrices to the well-known Zachary's karate club network (Zachary, 1977) as shown in Figure 1a, which contains 34 nodes of 2 classes and 78 unweighted edges (grey solid lines). We then increase the weights of intra-class edges by ten times to simulate GCN-LPA. We find that GCN works well on this network (Figure 1b), but GCN-LPA performs even better than GCN because the node embeddings are completely linearly separable as shown in Figure 1c. To further justify our claim, we randomly add 20 "noisy" inter-class edges (grey dotted lines) to the original network, from which we observe that GCN is misled by noise and mixes nodes of two classes together (Figure 1d), but GCN-LPA still distinguishes the two clusters (Figure 1e) because it is better at "denoising" undesirable edges based on the supervised signal of labels.

3 Connection to Existing Work

Edge weights play a key role in graph-based node classification as well as representation learning. In this section, we discuss three lines of related work that learn edge weights adaptively.

Locally linear embedding (LLE) (Roweis & Saul, 2000) and its variants (Zhang & Wang, 2007; Kong et al., 2012) learn edge weights by constructing a linear dependency between a node and its neighbors, 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.

Classical LPA (Zhu et al., 2005; Zhou et al., 2004) 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 (Zhu et al., 2003; Liu et al., 2019a) (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 (Wang & Zhang, 2008; Karasuyama & Mamitsuka, 2013), using leave-one-out loss (Zhang & Lee, 2007), or imposing sparseness on edge weights

(Hong et al., 2009). 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.

Our method is also conceptually connected to **attention mechanisms on graphs** (Veličković et al., 2018; Thekumparampil et al., 2018; Zhang et al., 2018; Liu et al., 2019b), 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 (Veličković et al., 2018), $\alpha_{ij} = a \cdot \cos(W\mathbf{x}_i, W\mathbf{x}_j)$ in AGNN (Thekumparampil et al., 2018), $\alpha_{ij} = (W_1\mathbf{x}_i)^\top W_2\mathbf{x}_j$ in GaAN (Zhang et al., 2018), and $\alpha_{ij} = \boldsymbol{a}^\top \tanh(W_1\mathbf{x}_i + W_2\mathbf{x}_j)$ in GeniePath (Liu et al., 2019b), where a and W are trainable variables. A significant difference between these attention mechanisms and our work is that attention weights are learned based merely on feature similarity, while we propose that edge weights should be consistent with the distribution of labels on the graph, which requires less handcrafting of the attention function and is more task-oriented. Nevertheless, all the above formulas for calculating attentions can also be used in our model as the implementation of edge weights.

4 Experiments

We evaluate our model and present its performance on five datasets including citation networks and coauthor networks. We also study the hyper-parameter sensitivity and provide training time analysis.

4.1 Datasets

We use the following five datasets in our experiments:

Citation networks: We consider three citation network datasets (Sen et al., 2008): **Cora**, **Citeseer**, and **Pubmed**. In these datasets, 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. **Coauthor networks**: We also use two co-authorship networks (Shchur et al., 2018), **Coauthor-CS** and **Coauthor-Phy**, based on Microsoft Academic Graph from the KDD Cup 2016 challenge. Here 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 2.5 that node classification benefits from intra-class edges in a graph.

4.2 Baselines

We compare against the following baselines in our experiments: **Multi-layer Perceptron (MLP)** and **Logistic Regression (LR)** are feature-based methods that do not consider the graph structure. **Label Propagation (LPA)** (Zhu et al., 2005), on the other hand, only consider the graph structure and ignore node features. The rest of baselines are GCN-based methods: **Graph Convolutional Network (GCN)** (Kipf & Welling, 2017) proposes a first-order approximation to spectral graph convolutions. **Graph Attention Network (GAT)** (Veličković et al., 2018) propose an attention mechanism to treat neighbors differently in the aggregation step. **Jumping Knowledge Networks (JK-Net)** (Xu et al., 2018) leverages different neighborhood ranges for each node to en-

	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
# features	1,433	3,703	500	6,805	8,415
# classes	7	6	3	15	5
Intra-class edge rate	81.0%	73.6%	80.2%	80.8%	93.1%

Table 1: Dataset statistics after removing self-loops and duplicate edges.

Method	**Cora**	**Citeseer**	**Pubmed**	**Coauthor-CS**	**Coauthor-Phy**
MLP	64.6 ± 1.7	62.0 ± 1.8	85.9 ± 0.3	91.7 ± 1.4	94.1 ± 1.2
LR	77.3 ± 1.8	71.2 ± 1.8	86.0 ± 0.6	91.1 ± 0.6	93.8 ± 1.1
LPA	85.3 ± 0.9	70.0 ± 1.7	82.6 ± 0.6	91.3 ± 0.2	94.9 ± 0.4
GCN	88.2 ± 0.8	77.3 ± 1.5	87.2 ± 0.4	93.6 ± 1.5	96.2 ± 0.2
GAT	87.7 ± 0.3	76.2 ± 0.9	86.9 ± 0.5	93.8 ± 0.4	96.3 ± 0.7
JK-Net	**89.1** ± 1.2	78.3 ± 0.9	85.8 ± 1.1	92.4 ± 0.4	94.8 ± 0.4
GraphSAGE	86.8 ± 1.9	75.2 ± 1.1	84.7 ± 1.6	92.6 ± 1.6	94.5 ± 1.1
GCN-LPA	88.5 ± 1.5	**78.7** ± 0.6	**87.8** ± 0.6	**94.8** ± 0.4	**96.9** ± 0.2

Table 2: Mean and the 95% confidence intervals of test set accuracy for all methods and datasets.

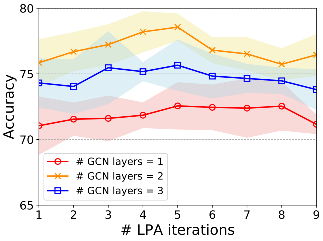


Figure 2: Sensitivity to # LPA iterations on Citeseer dataset.

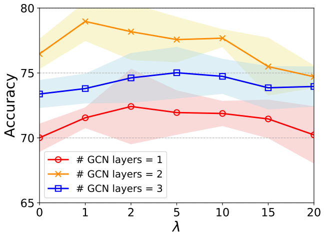


Figure 3: Sensitivity to λ on Citeseer dataset.

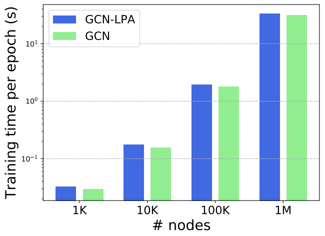


Figure 4: Training time per epoch on random graphs.

able structure-aware representation. We use concat as the aggregator for JK-Net. **Graph Sampling and Aggregation (GraphSAGE)** (Hamilton et al., 2017) is a mini-batch implementation of GCN that uses neighborhood sampling strategy and different aggregation schemes. We use mean as the aggregator for GraphSAGE.

4.3 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] The ratio of training, validation, and test set are set as $6 : 2 : 2$. The weight of each edge is treated as a free variable during training. We train our model for 200 epochs using Adam (Kingma & Ba, 2015) and report the test set accuracy when validation set accuracy is maximized. Each experiment is repeated three times and we report the mean and the 95% confidence interval. We initialize weights according to Glorot & Bengio (2010) and row-normalize input features. During training, we apply L2 regularization to the transformation matrices and use the dropout technique (Srivastava et al., 2014). The settings of all other hyper-parameters can be found in Appendix F.

4.4 Results

The results of node classification are summarized in Table 2. Table 2 indicates that only using node features (MLP, LR) or graph structure (LPA) will lead to information loss and cannot fully exploit datasets in general. The results demonstrate that our proposed GCN-LPA model surpasses state-of-the-art GCN/GNN baselines. We note that JK-Net is a strong baseline on Cora, but it does not perform consistently well on other datasets.

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 2 and 3, respectively, where each line corresponds to a given number of GCN layers in GCN-LPA. From Figure 2 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 3 shows that training without the LPA loss term (i.e., $\lambda = 0$) is more difficult than the case where $\lambda = 1 \sim 5$, which justifies our aforementioned claim that it is

[3] The experimental setting here is the same as GCN (Kipf & Welling, 2017). But note that our method can be easily generalized to inductive case if implemented in a way similar to GraphSAGE (Hamilton et al., 2017).

Ratio of labeled nodes	0%	20%	40%	60%	80%	100%
Accuracy	75.8 ± 1.0	76.3 ± 1.1	76.7 ± 0.8	77.3 ± 0.7	78.1 ± 0.6	78.7 ± 0.6

Table 3: Result of GCN-LPA on Citeseer dataset with differet ratio of labeled nodes in LPA.

hard for the GCN part to learn both transformation matrices W and edge weights A simultaneously without the assistance of LPA regularization.

To further show how much the LPA impacts the performance, we vary the ratio of labeled nodes in LPA from 100% to 0% during training, and report the result of acuracy on Citeseer dataset in Table 3. From Table 3 we observe that the performance of GCN-LPA gets worse when the ratio of labeled nodes in LPA decreases. In addition, using more labeled nodes in LPA also helps improve the model stability. Note that a ratio of 0% does not mean that GCN-LPA is equivalent to GCN (Kipf & Welling, 2017) because the edge weights in GCN-LPA is still trainable, which increases the risk of overfitting the training data.

We study the training time of GCN-LPA on random graphs. We use the one-hot identity vector as feature and 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 set as 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 @2.60GHz), and 128GB 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 4. Our result shows that GCN-LPA requires only 9.2% extra training time on average compared to GCN.

5 Conclusion and Future Work

In this paper, we studied the theoretical relationship between two types of well-known graph-based models for node classification, label propagation algorithm and graph convolutional neural networks, from the perspectives of feature/label smoothing and 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 in node classification. Experiments on five datasets 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 avenues of possible directions for future work. First, our proposed model focuses on transductive setting where all node features and the entire graph structure are given. An interesting problem is whether it can be applied to inductive setting where we have no access to test nodes during training. Second, the question of how to generalize the idea of our model to GNNs with different aggregation functions (e.g., concatenation or max-pooling) is also a promising direction.

References

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

Chen Gong, Dacheng Tao, Wei Liu, Liu Liu, and Jie Yang. Label propagation via teaching-to-learn and learning-to-teach. *IEEE Transactions on Neural Networks and Learning Lystems*, 28 (6), 2017.

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

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

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

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

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

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

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

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

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

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

Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. Geniepath: Graph neural networks with adaptive receptive paths. In *The 33rd AAAI Conference on Artificial Intelligence*, 2019b.

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

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

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

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

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1), 2014.

Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

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

Fei Wang and Changshui Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1), 2008.

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

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

Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4), 1977.

Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294*, 2018.

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

Zhenyue Zhang and Jing Wang. Mlle: Modified locally linear embedding using multiple weights. In *Advances in Neural Information Processing Systems*, 2007.

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

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

Xiaojin Zhu, John Lafferty, and Ronald Rosenfeld. *Semi-supervised learning with graphs*. PhD thesis, Carnegie Mellon University, language technologies institute, school of , 2005.

APPENDIX

A PROOF OF THEOREM 1

Proof. Denote $\tilde{a}_{ij} = a_{ij}/d_{ii}$ as the normalized weight of edge (j, i). It is clear that $\sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij} = 1$. Given that $\mathcal{M}$ is differentiable, we perform a first-order Taylor expansion with Peano's form of remainder at $\mathbf{x}_i$ for $\sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij} y_j$:

$$\begin{aligned}
&\sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij} y_j = \sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij}\mathcal{M}(\mathbf{x}_j) \\
&= \sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij}\left(\mathcal{M}(\mathbf{x}_i) + \frac{\partial\mathcal{M}(\mathbf{x}_i)}{\partial\mathbf{x}^\top}(\mathbf{x}_j - \mathbf{x}_i) + o(\|\mathbf{x}_j - \mathbf{x}_i\|_2)\right) \\
&= \mathcal{M}(\mathbf{x}_i) + \frac{\partial\mathcal{M}(\mathbf{x}_i)}{\partial\mathbf{x}^\top}\sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij}(\mathbf{x}_j - \mathbf{x}_i) + \sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij} o(\|\mathbf{x}_j - \mathbf{x}_i\|_2) \\
&= y_i - \frac{\partial\mathcal{M}(\mathbf{x}_i)}{\partial\mathbf{x}^\top}\epsilon_i + \sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij} o(\|\mathbf{x}_j - \mathbf{x}_i\|_2).
\end{aligned} \tag{10}$$

According to Cauchy-Schwarz inequality and L-Lipschitz property, we have

$$\left|\frac{\partial\mathcal{M}(\mathbf{x}_i)}{\partial\mathbf{x}^\top}\epsilon_i\right| \le \left\|\frac{\partial\mathcal{M}(\mathbf{x}_i)}{\partial\mathbf{x}^\top}\right\|_2 \|\epsilon_i\|_2 \le L\|\epsilon_i\|_2. \tag{11}$$

Therefore, the approximation of y_i is bounded by

$$\begin{aligned}
&\left|y_i - \sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij} y_j\right| \\
&= \left|\frac{\partial\mathcal{M}(\mathbf{x}_i)}{\partial\mathbf{x}^\top}\epsilon_i - \sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij} o(\|\mathbf{x}_j - \mathbf{x}_i\|_2)\right| \\
&\le \left|\frac{\partial\mathcal{M}(\mathbf{x}_i)}{\partial\mathbf{x}^\top}\epsilon_i\right| + \left|\sum_{j\in\mathcal{N}(i)} \tilde{a}_{ij} o(\|\mathbf{x}_j - \mathbf{x}_i\|_2)\right| \\
&\le L\|\epsilon_i\|_2 + o\big(\max_{j\in\mathcal{N}(i)}(\|\mathbf{x}_j - \mathbf{x}_i\|_2)\big).
\end{aligned} \tag{12}$$

□

B PROOF OF THEOREM 2

Before proving Theorem 2, we first give two lemmas that demonstrate the exact form of feature influence and label influence defined in this paper. The relationship between feature influence and label influence can then be deduced from their exact forms.

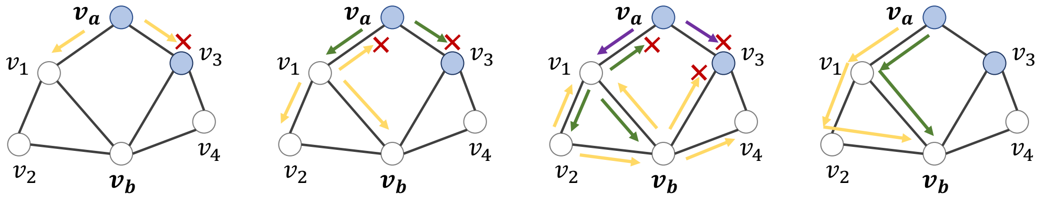
Lemma 1 *Assume that the nonlinear activation function in GCN is ReLU. Let $\mathcal{P}_k^{a\to b}$ be a path $[v^{(k)}, v^{(k-1)}, \cdots, v^{(0)}]$ of length k from node v_a to node v_b, where $v^{(k)} = v_a$, $v^{(0)} = v_b$, and $v^{(i-1)} \in \mathcal{N}(v^{(i)})$ for $i = k, \cdots, 1$. Then we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

$$\tilde{I}_f(v_a, v_b; k) = \sum_{\mathcal{P}_k^{a\to b}} \prod_{i=k}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}, \tag{13}$$

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

Proof. See Xu et al. (2018) for the detailed proof. □

The product term in Eq. (13) is the probability of a given path $\mathcal{P}_k^{a\to b}$. Therefore, the right hand side in Eq. (13) 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.



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3 (d) Paths from v_a to v_b

Figure 5: 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) The propagating process 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.

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

$$I_l(v_a, v_b; k) = \sum_{j=1}^{k} \sum_{\mathcal{U}_j^{a\to b}} \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}, \tag{14}$$

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: (1) 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. (2) 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 5 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 $j = 1, \cdots, k-1$. Therefore, the influence of y_b on $y_a^{(k)}$ is the cumulative influence of $y_b^{(j)}$ on $y_a^{(k)}$ for $j = 0, 1, \cdots, k-1$:

$$I_l(v_a, v_b; k) = \frac{\partial y_a^{(k)}}{\partial y_b} = \sum_{j=0}^{k-1} \frac{\partial y_a^{(k)}}{\partial y_b^{(j)}}. \tag{15}$$

According to the updating rule of LPA, we have

$$\frac{\partial y_a^{(k)}}{\partial y_b^{(j)}} = \frac{\partial \sum_{v_z \in \mathcal{N}(v_a)} \tilde{a}_{az} y_z^{(k-1)}}{\partial y_b^{(j)}} = \sum_{v_z \in \mathcal{N}(v_a)} \tilde{a}_{az} \frac{\partial y_z^{(k-1)}}{\partial y_b^{(j)}}. \tag{16}$$

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:

$$\frac{\partial y_a^{(k)}}{\partial y_b^{(j)}} = \sum_{v_z \in \mathcal{N}(v_a), z>m} \tilde{a}_{az} \frac{\partial y_z^{(k-1)}}{\partial y_b^{(j)}}, \tag{17}$$

where $z > m$ means v_z is unlabeled. To intuitively understand Eq. (17), 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 Eq. (17) in the way similar to what we do for $y_a^{(k)}$ in Eq. (16). So the expansion in Eq. (17) 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:

$$\frac{\partial y_a^{(k)}}{\partial y_b^{(j)}} = \sum_{v_z \in \mathcal{V}} \sum_{\mathcal{U}_{k-j}^{a \to z}} \left(\prod_{i=k-j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}} \right) \frac{\partial y_z^{(j)}}{\partial y_b^{(j)}}, \tag{18}$$

where v_z in the first summation term is the end node of a random walk, $\mathcal{U}_{k-j}^{a \to z}$ 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 $\mathcal{U}_{k-j}^{a \to z}$. Consider the last term $\frac{\partial y_z^{(j)}}{\partial y_b^{(j)}}$ in Eq. (18). 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 Eq. (18). Therefore, we have

$$\frac{\partial y_a^{(k)}}{\partial y_b^{(j)}} = \sum_{\mathcal{U}_{k-j}^{a \to b}} \prod_{i=k-j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}, \tag{19}$$

where $\mathcal{U}_{k-j}^{a \to b}$ 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 Eq. (15) with Eq. (19), we obtain that

$$I_l(v_a, v_b; k) = \sum_{j=0}^{k-1} \sum_{\mathcal{U}_{k-j}^{a \to b}} \prod_{i=k-j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}} = \sum_{j=1}^{k} \sum_{\mathcal{U}_{j}^{a \to b}} \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}. \tag{20}$$

□

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

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

$$\begin{aligned}
\mathbb{E}\big[I_l(v_a, v_b; k)\big] =& \mathbb{E}\left[\sum_{j=1}^{k} \sum_{\mathcal{U}_j^{a \to b}} \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}} \right] = \sum_{j=1}^{k} \mathbb{E}\left[\sum_{\mathcal{U}_j^{a \to b}} \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}} \right] \\
=& \sum_{j=1}^{k} \sum_{\mathcal{P}_j^{a \to b}} \Pr\left(\mathcal{P}_j^{a \to b} \text{ is an unlabeled-nodes-only path}\right) \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}} \\
=& \sum_{j=1}^{k} \sum_{\mathcal{P}_j^{a \to b}} \beta^j \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}} \\
=& \sum_{j=1}^{k} \beta^j \tilde{I}_f(v_a, v_b; j).
\end{aligned} \tag{21}$$

□

C Proof of Theorem 3

Proof. Denote the set of labels as $\mathcal{L}$. Since different label dimensions in $y_a^{(\cdot)}$ do not interact with each other when running LPA, the value of the y_a-th 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

$$y_a^{(k)}[y_a] = \sum_{v_b : y_b = y_a} \sum_{j=1}^{k} \sum_{\mathcal{U}_j^{a \to b}} \prod_{i=j}^{1} \tilde{a}_{v^{(i-1)}, v^{(i)}}, \tag{22}$$

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

$$\Pr(\hat{y}_a = y_a) = \frac{y_a^{(k)}[y_a]}{\sum_{i \in \mathcal{L}} y_a^{(k)}[i]} \propto y_a^{(k)}[y_a] = \sum_{v_b : y_b = y_a} I_l(v_a, v_b; k) \tag{23}$$

□

D Proof of Theorem 4

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

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 $D(\mathbf{x}) = \frac{1}{2} \sum_{v_i, v_j} \tilde{a}_{ij} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$:

$$\nabla^2 D(\mathbf{x}) = \begin{bmatrix} 1 - \tilde{a}_{11} & -\tilde{a}_{12} & \cdots & -\tilde{a}_{1n} \\ -\tilde{a}_{21} & 1 - \tilde{a}_{22} & \cdots & -\tilde{a}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\tilde{a}_{n1} & -\tilde{a}_{n2} & \cdots & 1 - \tilde{a}_{nn} \end{bmatrix} = I - D^{-1}A. \tag{24}$$

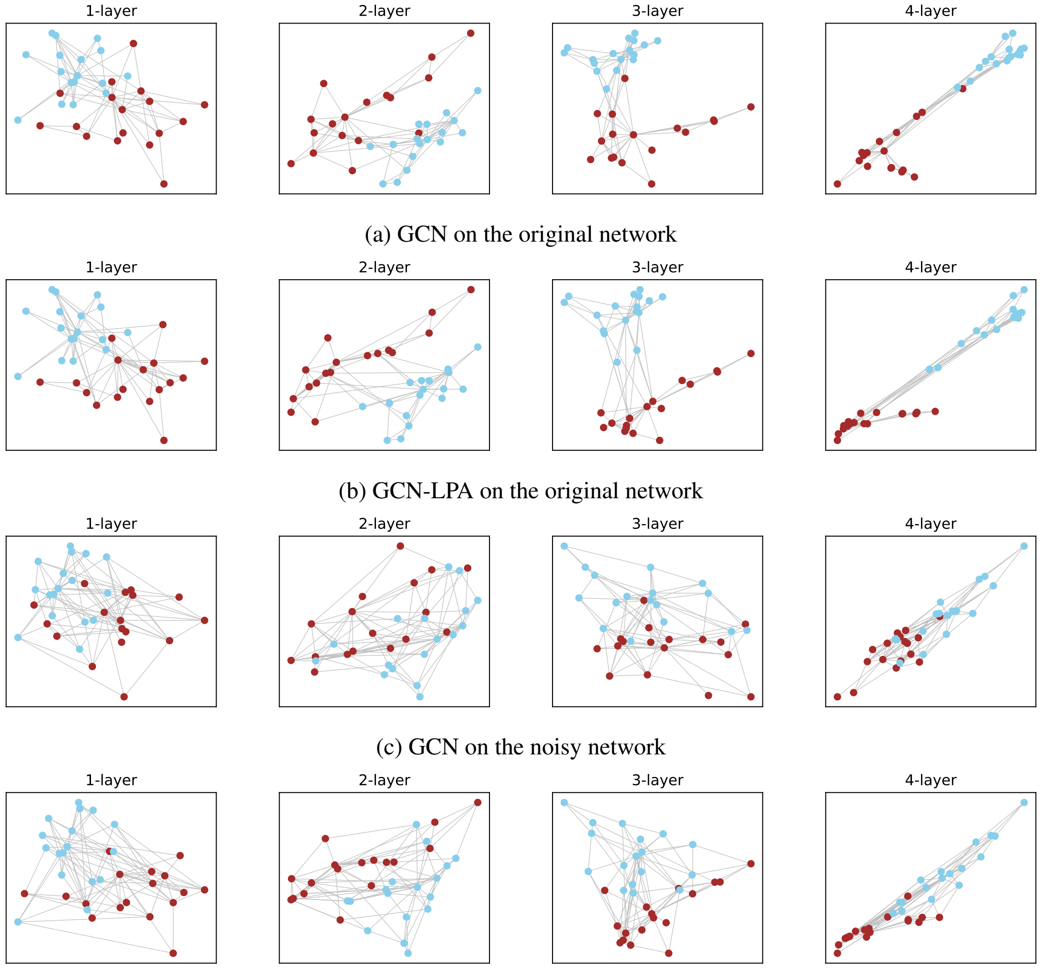
Therefore, $2I - \nabla^2 D(\mathbf{x}) = I + D^{-1}A$. Since $D^{-1}A$ is Markov matrix (i.e., each entry is non-negative and the sum of each row is one), its eigenvalues are within the range [-1, 1], so the eigenvalues of $I + D^{-1}A$ are within the range [0, 2]. Therefore, $I + D^{-1}A$ is a positive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

We can now prove Theorem 4:

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:

$$\begin{aligned} D(\mathbf{h}^{(k)}) =& D(\mathbf{x}^{(k)}) + \nabla D(\mathbf{x}^{(k)})^\top (\mathbf{h}^{(k)} - \mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{h}^{(k)} - \mathbf{x}^{(k)})^\top \nabla^2 D(\mathbf{x})(\mathbf{h}^{(k)} - \mathbf{x}^{(k)}) \\ =& D(\mathbf{x}^{(k)}) - \nabla D(\mathbf{x}^{(k)})^\top \nabla D(\mathbf{x}^{(k)}) + \frac{1}{2} \nabla D(\mathbf{x}^{(k)})^\top \nabla^2 D(\mathbf{x}) \nabla D(\mathbf{x}^{(k)}) \\ \leq& D(\mathbf{x}^{(k)}) - \nabla D(\mathbf{x}^{(k)})^\top \nabla D(\mathbf{x}^{(k)}) + \nabla D(\mathbf{x}^{(k)})^\top \nabla D(\mathbf{x}^{(k)}) = D(\mathbf{x}^{(k)}). \end{aligned} \tag{25}$$

□



(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

Figure 6: Visualization of GCN and GCN-LPA with $1 \sim 4$ layers on karate club network.

E More Visualization Results on Karate Club Network

Figure 6 illustrates more visualization of GCN and GCN-LPA on karate club network. In each subfigure, we vary the number of layers from 1 to 4 to examine how the learned representations evolve. The initial node features are one-hot identity vectors, and 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 Figure 6a and 6c (or Figure 6b and 6d), we conclude that more inter-class edges will make the separation harder for GCN (or GCN-LPA). Comparing Figure 6a and 6b (or Figure 6c and 6d), we conclude that GCN-LPA is more noise-resistant than GCN, therefore, GCN-LPA can better differentiate classes and identify clustering substructures.

F Hyper-parameter Settings

The detailed hyper-parameter settings for all datasets are listed in Table 4. In GCN-LPA, we use the same dimension for all hidden layers. 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 perform fine-tuning on

	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
L2 weight	1×10^{-4}	5×10^{-4}	2×10^{-4}	1×10^{-4}	1×10^{-4}
LPA weight (λ)	10	1	1	2	1
Dropout rate	0.2	0	0	0.2	0.2
Learning rate	0.05	0.2	0.1	0.1	0.05

Table 4: Hyper-parameter settings for all datasets.

other datasets, i.e., varying one hyper-parameter per time to see if the performance can be further improved. The search spaces for hyper-parameters are as follows:

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