# On the Equivalence of Decoupled Graph Convolution Network and Label Propagation

Hande Dong[1], Jiawei Chen[1*], Fuli Feng[2], Xiangnan He[1], Shuxian Bi[1], Zhaolin Ding[3], Peng Cui[4]
[1]University of Science and Technology of China, [2]National University of Singapore,
[3]North Carolina State University, [4]Tsinghua University.
donghd@mail.ustc.edu.cn,cjwustc@ustc.edu.cn,fulifeng93@gmail.com,xiangnanhe@gmail.com,
stanbi@mail.ustc.edu.cn,zding8@ncsu.edu,cuip@tsinghua.edu.cn

## ABSTRACT

The original design of Graph Convolution Network (GCN) couples *feature transformation* and *neighborhood aggregation* for node representation learning. Recently, some work shows that coupling is inferior to decoupling, which supports deep graph propagation better and has become the latest paradigm of GCN (e.g., APPNP [16] and SGCN [32]). Despite effectiveness, the working mechanisms of the decoupled GCN are not well understood.

In this paper, we explore the decoupled GCN for semi-supervised node classification from a novel and fundamental perspective — label propagation. We conduct thorough theoretical analyses, proving 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. More interestingly, we reveal the effectiveness of decoupled GCN: going beyond the conventional label propagation, it could automatically assign structure- and model- aware weights to the pseudo-label data. This explains why the decoupled GCN is relatively robust to the structure noise and over-smoothing, but sensitive to the label noise and model initialization. Based on this insight, we propose a new label propagation method named Propagation then Training Adaptively (PTA), which overcomes the flaws of the decoupled GCN with a dynamic and adaptive weighting strategy. Our PTA is simple yet more effective and robust than decoupled GCN. We empirically validate our findings on four benchmark datasets, demonstrating the advantages of our method. The code is available at https://github.com/DongHande/PT_propagation_then_training.

## CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → *Neural networks.*

## KEYWORDS

Graph Convolution Network, Graph Neural Networks, Decoupled Graph Neural Network, Label Propagation

∗ Jiawei Chen is the corresponding author.


*WWW '21, April 19–23, 2021, Ljubljana, Slovenia*

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449927

**ACM Reference Format:**
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 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3449927

## 1 INTRODUCTION

Graphs, which reflect relationships between entities, are ubiquitous in the real world, such as social, citation, molecules and biological networks. Recent years have witnessed the flourish of deep learning approaches on graph-based applications, such as node classification [5, 18], graph classification [17, 39], link prediction [11, 41], and community detection [13]. Among various techniques, Graph Convolutional Network (GCN) has drawn recent attention due to its effectiveness and flexibility [7, 8, 15, 30, 36, 40, 44].

There are two important operations in a spatial GCN model: 1) *feature transformation*, which is inherited from conventional neural networks to learn node representations from the past features, and 2) *neighborhood aggregation* (also termed as *propagation*), which updates the representation of a node by aggregating the representations of its neighbors. In the original GCN [15] and many follow-up models [30, 35, 45], the two operations are coupled, i.e., each graph convolution layer is consisted of both feature transformation and neighborhood aggregation. Nevertheless, some recent work find that such a coupling design is unnecessary and causes many issues such as training difficulty, hard to leverage graph structure deeply, and over-smoothing [11, 16, 32]. By separating the two operations, simpler yet more effective and interpretable models can be achieved, like the APPNP [16], SGCN [32], DAGNN [19] for node classification, and LightGCN [11] for link prediction. We term these models that separate the neural network from the propagation scheme as *decoupled GCN*, which has become the latest paradigm of GCN. Despite effectiveness, the working mechanisms of the decoupled GCN are not well understood.

In this work, we strive to analyze the decoupled GCN deeply and provide insights into how it works for node classification. We prove in theory (by comparing the gradients) that the training stage of the decoupled GCN is essentially equivalent to performing a two-step label propagation. Specifically, the first step propagates the known labels along the graph to generate pseudo-labels for the unlabeled nodes, and then the second step trains (non-graph) neural network predictor on the augmented pseudo-labeled data. This novel view of label propagation reveals the reasons of the effectiveness of decoupled GCN:

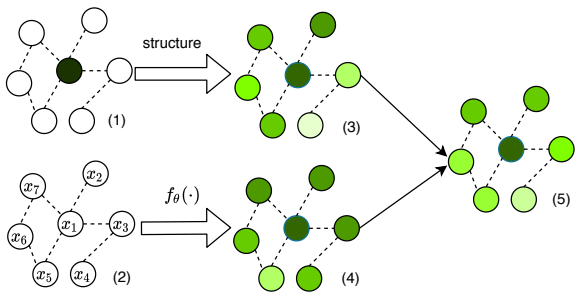


**Figure 1: An illustration of the equivalent label propagation view of the decoupled GCN. The dark node in (1) is a labeled source node in the training set. The first step propagates the label of this source node to its $K$-hop neighbors, *i.e.,* pseudo-labeling neighbor nodes, and the second step uses the augmented data to train the neural network predictor. The color depth represents the weights of the pseudo-labeled data. The final weights of pseudo-labels in (5) is determined by both graph structure (1)->(3) and model prediction (2)->(4).**

(1) The pseudo-labeled data serves as an augmentation to supplement the input labeled data. In semi-supervised graph learning settings, the labeled data is usually of small quantity, making neural network predictors have a large variance and easy to overfit. As such, the pseudo-labeled data augmentation helps to reduce overfitting and improve the generalization ability.

(2) Instead of assigning a uniform weight to the pseudo-labeled data, decoupled GCN dynamically adjusts the weights based on the graph structure and model prediction during training. It follows two intuitions to adjust the weight of an unlabeled node: first, the node that is closer to the labeled source node is given a larger weight, and second, the node that has the pseudo-label different from the prediction of the neural network is given a smaller weight. The two intuitions are reasonable and explain why decoupled GCN is relatively robust to structure noise and over-smoothing.

(3) To predict the label of a node in the inference stage, decoupled GCN combines the predictions of the node's $k$-hop neighbors rather than basing on the node's own prediction. Given the local homogeneity assumption of a graph, such an ensemble method effectively reduces the prediction variance, improving the accuracy and robustness of model prediction.

Figure 1 illustrates the process. In addition to the advantages revealed by the equivalent label propagation view, we further identify two limitations of existing decoupled GCN:

(1) High sensitivity to model initialization. Since the weights of pseudo-labeled data are dynamically adjusted based on model prediction, the initialization of the model exerts a much larger impact on the model training. An ill initialization would generate incorrect predictions in the beginning, making the weights of pseudo-labeled data diverge from the proper values. As a result, the model may converge to an undesirable state.

(2) Lacking robustness to label noise. The weights of the pseudo-labeled data generated from a labeled source node are normalized to be unity. It implies that different labeled nodes are assumed to have an equal contribution to weigh the pseudo-labeled data. Such an assumption ignores the quality or importance of labeled data, which may not be ideally clean and have certain noises in practical applications.

Our analyses provide deep insights into how the decoupled GCN works or fails, inspiring us to develop a better method by fostering strengths and circumventing weaknesses. Specifically, we propose a new method named *Propagation then Training Adaptively* (PTA), which augments the classical label propagation with a carefully designed adaptive weighting strategy. After generating the pseudo-labeled data with label propagation, we dynamically control their weights during modeling training. Firstly, PTA abandons the equal-contribution assumption on labeled nodes, specifying the importance of a labeled node based on the consistence between its label and the predicted labels of its neighbors. Secondly, in the early stage of training, PTA reduces the impact of model prediction on the weighting of pseudo-labeled data, since an immature neural network usually yields unreliable prediction; as the training proceeds, PTA gradually enlarges the impact to increase the model's robustness to noise. Through the two designs, we eliminate the limitations of decoupled GCN and meanwhile make full use of its advantages.

We summarize the contributions of this paper as below:

- Conducting thorough theoretical analyses of decoupled GCN, proving that its training stage is equivalent to performing a two-step label propagation.
- Analyzing the advantages and limitations of decoupled GCN from the label propagation perspective.
- Proposing a new label propagation method PTA that inherits the merits of decoupled GCN and overcomes its limitations with a carefully designed weighting strategy for pseudo-labeled data.
- Validating our findings on four datasets of semi-supervised node classification and demonstrating the superiority of PTA in multiple aspects of accuracy, robustness, and stability.

## 2 PRELIMINARIES

In this section, we first formulate the node classification problem (Section 2.1). We then briefly introduce GCN (Section 2.2) and provide a summary of existing decoupled GCN (Section 2.3). Finally the classical label propagation is presented (Section 2.4).

### 2.1 Problem Formulation

Suppose we have a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the node set with $|\mathcal{V}| = n$ and $\mathcal{E}$ is the edge set. In this paper, we consider an undirected graph, whose adjacency matrix is represented with a matrix $\boldsymbol{A} \in \mathsf{R}^{n\times n}$. Let $\boldsymbol{D} = diag(d_1, d_2, ..., d_n)$ denote the degree matrix of $\boldsymbol{A}$, where $d_i = \sum_{j\in\mathcal{V}} a_{ij}$ is the degree of the node $i$. The normalized adjacency matrix is represented as $\hat{\boldsymbol{A}}$. There are several normalization strategies, *e.g.,* $\hat{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ [2], or $\hat{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A}$ [42], or $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ [15] where self-loop has been added. Also, we have features of the nodes, which are represented as a matrix $\boldsymbol{X} \in \mathsf{R}^{n\times f}$. In this paper, we aim to solve graph-based semi-supervised node classification. Suppose that only a small set of nodes $i \in \mathcal{V}_l \subseteq \mathcal{V}$ have labels $h(i) \in C$ from the label set $C = \{1, 2, ..., C\}$. We also write the label $h(i)$ as a one-hot vector

**Table 1: Notation and Definitions.**

| Notation | Annotation |
|---|---|
| $\hat{\boldsymbol{A}}$ | the normalized adjacency matrix |
| $\bar{\boldsymbol{A}}$ | $\sum \beta_k \hat{\boldsymbol{A}}^k$, the combination of $\hat{\boldsymbol{A}}^k$, $k = 0, 1, 2 \cdots$ |
| $\boldsymbol{X} \in \mathsf{R}^{n \times f}$ | the feature matrix of the nodes |
| $\mathcal{V}_l, \mathcal{V}, C$ | the labeled node set, the universal node set, the label set |
| $h(i) \in C$ | the label id of node $i$ |
| $\boldsymbol{y}_i \in \mathsf{R}^C$ | the one-hot representation of the label of node $i$ |
| $Y$ | the observed label matrix |
| $Y_{soft}$ | the soft label matrix using label propagation |

$\boldsymbol{y}_i \in \mathsf{R}^C$. Let $\boldsymbol{Y}$ denotes the observed label matrix, where $i$-th row is either set as $\boldsymbol{y}_i$ for $i \in \mathcal{V}_l$ or a zero vector otherwise. The goal is to predict the labels for the unlabeled nodes.

## 2.2 Graph convolution network (GCN)

GCN performs layer-wise feature transformation and neighborhood aggregation. Each layer of GCN can be written as:

$$\boldsymbol{H}^{(k+1)} = \sigma\left(\hat{\boldsymbol{A}}\boldsymbol{H}^{(k)}\boldsymbol{W}^{(k)}\right), \tag{1}$$

where $\boldsymbol{H}^{(0)} = \boldsymbol{X}$, $\sigma(.)$ is an activation function such as ReLU; $\boldsymbol{H}^{(k)}$ and $\boldsymbol{W}^{(k)}$ denote the node representations and transformation parameters of the $k$-th layer. As we can see, each layer consists of two important operations: 1) neighborhood aggregation $\boldsymbol{P}^{(k)} = \hat{\boldsymbol{A}}\boldsymbol{H}^{(k)}$: which refines the representation of a node by aggregating the representations of its neighbors; 2) feature transformation $\boldsymbol{H}^{(k+1)} = \sigma(\boldsymbol{P}^{(k)}\boldsymbol{W}^{(k)})$, which is inherited from conventional neural networks to perform nonlinear feature transformation. By stacking multiple layers, GCN generates node representations by integrating both node features and the graph structure.

## 2.3 Decoupled GCN

In this subsection, we first define a general architecture of decoupled GCN and show how it subsumes specific decoupled models [11, 16, 19, 32]. We then transform it into a more concise formulation to facilitate theoretically analyzing it in the next section.

In the original GCN [15] and many follow-up models [30, 35, 45], neighborhood aggregation and feature transformation are coupled with each layer. Nevertheless, some recent works find that such a coupling design is unnecessary and propose to separate these two operations. These models can be summarized to a general architecture named *decoupled GCN*:

$$\hat{\boldsymbol{Y}} = \text{softmax}\left(\bar{\boldsymbol{A}}\boldsymbol{f}_\theta(\boldsymbol{X})\right), \tag{2}$$

where $\boldsymbol{f}_\theta(\cdot)$ is a feature transformation function, which can be done by neural network. $\bar{\boldsymbol{A}} = \sum \beta_k \hat{\boldsymbol{A}}^k$ is determined by the graph structure and the propagation strategy, whose element reflects the proximity of two nodes in the graph. We will next show how this architecture subsumes existing decoupled methods.

*APPNP and DAGNN.* APPNP [16] claims several advantages of separating neural network modeling with the propagation scheme, formulating the model as:

$$\begin{aligned} \boldsymbol{H}^{(0)} &= \boldsymbol{f}_\theta(\boldsymbol{X}) \\ \boldsymbol{H}^{(k)} &= (1-\alpha)\hat{\boldsymbol{A}}\boldsymbol{H}^{(k-1)} + \alpha\boldsymbol{H}^{(0)}, \quad k = 1, 2, \ldots K-1 \\ \hat{\boldsymbol{Y}}_{APPNP} &= \text{softmax}\left(\boldsymbol{H}^{(K)}\right). \end{aligned} \tag{3}$$

APPNP uses Personalized PageRank [24] as the propagation strategy on graph. The model can be subsumed into Equation 2 by setting $\bar{\boldsymbol{A}} = (1-\alpha)^K\hat{\boldsymbol{A}}^K + \alpha\sum_{k=0}^{K-1}(1-\alpha)^k\hat{\boldsymbol{A}}^k$. The proof is presented in Appendix A. When $K \to \infty$, $\bar{\boldsymbol{A}} = \alpha(\boldsymbol{I} - (1-\alpha)\hat{\boldsymbol{A}})^{-1}$ is the diffusion kernel [43], which has been widely adopted to measure proximity in the graph. Another recent work DAGNN [19] is similar to APPNP but uses a different propagation scheme: $\bar{\boldsymbol{A}} = \sum_{k=0}^{K} s_k\hat{\boldsymbol{A}}^k$, where $s_k$ controls the importance of different layers.

*SGCN and LightGCN.* The two models are designed by simplifying the original GCN to make it more concise and appropriate for downstream applications. SGCN and LightGCN are similar but for different tasks, so here we just present SGCN:

$$\hat{\boldsymbol{Y}}_{\text{SGCN}} = \text{softmax}\left(\boldsymbol{S}^K\boldsymbol{X}\boldsymbol{\Theta}\right). \tag{4}$$

Naturally, it can be subsumed into Equation 2 by setting $\boldsymbol{f}_\theta(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{\Theta}$ and $\bar{\boldsymbol{A}} = \boldsymbol{S}^K$.

In this paper, we would like to define a more concise formulation of decoupled GCN as follows:

$$\hat{\boldsymbol{Y}} = \bar{\boldsymbol{A}}\boldsymbol{f}_\theta(\boldsymbol{X}), \tag{5}$$

where softmax function has been integrated into feature transformation function, *i.e.,* $\boldsymbol{f}_\theta(\boldsymbol{X}) \leftarrow \text{softmax}(\boldsymbol{f}_\theta(\boldsymbol{X}))$. Although the predictions are not probabilities, integrating softmax$(\cdot)$ into $\boldsymbol{f}_\theta$ does not affect model performance, since softmax$(\cdot)$ is a monotonic function. Moreover, we analyze such an operation does not affect the model optimization in Appendix B.

## 2.4 Label Propagation

Label propagation (LP) [34] is a classic semi-supervised learning algorithm that propagates the known labels along the graph to other unlabeled nodes. It can be formulated as follows:

$$\begin{aligned} \boldsymbol{Y}^{(0)} &= \boldsymbol{Y} \\ \boldsymbol{Y}^{(k)} &= \hat{\boldsymbol{A}}\boldsymbol{Y}^{(k-1)} \quad k = 1, 2, \ldots, K-1 \\ \boldsymbol{y}_i^{(k)} &= \boldsymbol{y}_i, \quad \forall i \in \mathcal{V}_l, \end{aligned} \tag{6}$$

where $\boldsymbol{Y}^{(k)}$ denotes the soft label matrix in iteration $k$, where each element $y_{ic}^{(k)}$ reflects how likely that the label of node $i$ is predicted as the label $c$. Similar to decoupled GCN, various propagation schemes can be adopted for LP, such as Personalized PageRank which has been adopted by APPNP where $\boldsymbol{Y}^{(k)} = (1-\alpha)\hat{\boldsymbol{A}}\boldsymbol{Y}^{(k-1)} + \alpha\boldsymbol{Y}^{(0)}$. Similar to Equation (5), the general formulation of LP can be summarized as follows:

$$\boldsymbol{Y}_{soft} = \hat{\boldsymbol{Y}} = \bar{\boldsymbol{A}}\boldsymbol{Y}. \tag{7}$$

# 3 UNDERSTANDING DECOUPLED GCN FROM LABEL PROROGATION

In this section, we first define a simple training framework called *Propagation then Training (PT)* (Section 3.1). We then conduct a

rigorous mathematical analysis of decoupled GCN to show that training a decoupled GCN is essentially equivalent to a case of PT (Section 3.2). Based on this insight, we further discuss the advantages and weaknesses of decoupled GCN (Section 3.3).

## 3.1 Propagation then Training

Before delving into decoupled GCN, we would like to design a simple model for node classification based on label propagation, which consists of two steps: 1) using the LP algorithm to propagate the known labels along the graph to generate pseudo-labels for the unlabeled nodes, and 2) training a neural network predictor on the augmented pseudo-labeled data. This training paradigm is simple and intuitive, using label propagation for data augmentation and benefiting model generalization. Formally, the model is optimized with the following objective function:

$$L(\theta) = \ell(\boldsymbol{f}_\theta(\boldsymbol{X}), \bar{\boldsymbol{A}}\boldsymbol{Y}), \tag{8}$$

where $\ell(\cdot)$ denotes the loss function between the predictions $\boldsymbol{f}_\theta(\boldsymbol{X})$ and the soft labels $\boldsymbol{Y}_{soft} = \bar{\boldsymbol{A}}\boldsymbol{Y}$. For the node classification task, cross-entropy loss is most widely used [22]. Then the objective function for Equation (8) can be re-written as follows:

$$\begin{aligned} L(\theta) &= -\sum_{i\in\mathcal{V}}\sum_{k\in C}\Big(\sum_{j\in\mathcal{V}}\bar{a}_{ij}y_{jk}\Big)\log f_{ik} \\ &= -\sum_{i\in\mathcal{V},j\in\mathcal{V}_l}\bar{a}_{ij}\sum_{k\in C}y_{jk}\log f_{ik} \\ &= \sum_{i\in\mathcal{V},j\in\mathcal{V}_l}\bar{a}_{ij}\,\mathrm{CE}\left(\boldsymbol{f}_i, \boldsymbol{y}_j\right), \end{aligned} \tag{9}$$

where $\bar{a}_{ij}$ denotes the $(i,j)$-th element of the matrix $\bar{\boldsymbol{A}}$, $\mathrm{CE}(\cdot)$ denotes the cross entropy loss between the predictions $\boldsymbol{f}_i$ and the pseudo-labels $\boldsymbol{y}_j$. $\boldsymbol{f}_i$ represents the result from the function $\boldsymbol{f}_\theta(\boldsymbol{x}_i)$ with features $\boldsymbol{x}_i$ for node $i$ and $\boldsymbol{f}_{i,k}$ denotes its $k-$th element. The reformulated loss can be interpreted as using node $j$'s label to train the node $i$ and weighting this pseudo-labeled instance $(\boldsymbol{x}_i, \boldsymbol{y}_j)$ based on their graph proximity $\bar{a}_{ij}$. It is reasonable as the closer nodes in the graph usually exhibit more similarity. Larger $\bar{a}_{ij}$ suggests larger likelihood that the node $j$ is labeled as $\boldsymbol{y}_i$. These augmented data transfer the knowledge of the labeled nodes to their close neighbors. The weight of instance $(\boldsymbol{x}_i, \boldsymbol{y}_j)$ is static $\bar{a}_{ij}$, *i.e.,* staying the same value during training, so we name this specific model as *Propagation then Training Statically (PTS)*.

In most cases, static weighting is unsatisfied (which will be discussed in next subsection). Thus, we would like to extend the PTS to a more general framework with flexible weighting strategies. We name the framework as *Propagation then Training (PT)*, which optimizes the following objective function:

$$L_{PT} = L(\theta) = \sum_{i\in\mathcal{V},j\in\mathcal{V}_l} w_{ij}\,\mathrm{CE}\left(\boldsymbol{f}_i, \boldsymbol{y}_j\right), \tag{10}$$

where $w \overset{\text{define}}{=} g(\boldsymbol{f}(X), \boldsymbol{A})$ means a general weighting strategy, which is controlled by the model prediction and propagation scheme with a specific function $g$.

## 3.2 Connection between Decoupled GCN and PT

In this subsection, we conduct a mathematical analysis of the gradients of decoupled GCN, proving that the training stage of decoupled GCN is essentially equivalent to a special case of PT. In fact, we have the following lemma:

Lemma 1. *Training a decoupled GCN is equivalent to performing Propagation then Training with dynamic weight* $w_{ij} = \frac{\bar{a}_{ji}f_{i,h(j)}}{\sum_{q\in\mathcal{V}}\bar{a}_{jq}f_{q,h(j)}}$ *for each pseudo-labeled data* $(\boldsymbol{x}_i, \boldsymbol{y}_j)$*, where* $h(j)$ *means the label ID of node* $j$*,* i.e., $y_{j,h(j)} = 1$.

Proof. The lemma can be proved by comparing the gradients of decoupled GCN and PT. Adopting cross entropy loss, the loss function of decoupled GCN is:

$$\begin{aligned} L_{DGCN} &= \ell(\bar{\boldsymbol{A}}\boldsymbol{f}_\theta(\boldsymbol{X}), Y) \\ &= -\sum_{j\in\mathcal{V}_l,k\in C} y_{jk}\Big(\log\sum_{i\in\mathcal{V}}\bar{a}_{ji}f_{ik}\Big). \end{aligned} \tag{11}$$

The gradients of the objective function *w.r.t.* $\theta$ can be written as:

$$\begin{aligned} \nabla_\theta L_{DGCN} &= -\sum_{j\in\mathcal{V}_l,k\in C} y_{jk}\nabla_\theta\Big(\log\sum_{i\in\mathcal{V}}\bar{a}_{ji}f_{ik}\Big) \\ &= -\sum_{j\in\mathcal{V}_l,k\in C} y_{jk}\frac{\sum_{i\in\mathcal{V}}\bar{a}_{ji}\nabla_\theta f_{ik}}{\sum_{q\in\mathcal{V}}\bar{a}_{jq}f_{qk}}. \end{aligned} \tag{12}$$

As $\boldsymbol{y}_j$ is an one-hot vector, only the $h(j)$-th element of $\boldsymbol{y}_j$ equal to one. The gradients can be rewritten as follows:

$$\begin{aligned} \nabla_\theta L_{DGCN} &= -\sum_{j\in\mathcal{V}_l} y_{j,h(j)}\frac{\sum_{i\in\mathcal{V}}\bar{a}_{ji}\nabla_\theta f_{i,h(j)}}{\sum_{q\in\mathcal{V}}\bar{a}_{jq}f_{q,h(j)}} \\ &= -\sum_{i\in\mathcal{V},j\in\mathcal{V}_l}\frac{\bar{a}_{ji}f_{i,h(j)}}{\sum_{q\in\mathcal{V}}\bar{a}_{jq}f_{q,h(j)}}y_{j,h(j)}\frac{\nabla_\theta f_{i,h(j)}}{f_{i,h(j)}} \\ &= \sum_{i\in\mathcal{V},j\in\mathcal{V}_l}\frac{\bar{a}_{ji}f_{i,h(j)}}{\sum_{q\in\mathcal{V}}\bar{a}_{jq}f_{q,h(j)}}\nabla_\theta\,\mathrm{CE}\left(\boldsymbol{f}_i, \boldsymbol{y}_j\right). \end{aligned} \tag{13}$$

Note that the gradients of the PT *w.r.t.* $\theta$ is:

$$\nabla_\theta L_{PT} = \sum_{j\in\mathcal{V}_l,i\in\mathcal{V}} w_{ij}\nabla_\theta\,\mathrm{CE}\left(\boldsymbol{f}_i, \boldsymbol{y}_j\right). \tag{14}$$

Comparing Equation (12) with Equation (14), the decoupled GCN is a special case of LP by setting $w_{ij}$ as $\frac{\bar{a}_{ji}f_{i,h(j)}}{\sum_{q\in\mathcal{V}}\bar{a}_{jq}f_{q,h(j)}}$.

## 3.3 Analyzing Decoupled GCN from PT

Based on above proof, the working mechanism of decoupled GCN can be well understood. It is equivalent to performing a label propagation to generate pseudo-labels and then optimizing the neural predictor on the pseudo-labeled data with a weighted loss function. As we can see from Equation (13), the essence of decoupled GCN is to construct more training instances with label propagation. Specifically, it propagates the known label of node $j$'s to other nodes (such as node $i$) and uses the augmented pseudo-label data $(\boldsymbol{x}_i, \boldsymbol{y}_j)$ to learn a better classifier. Moreover, these training instances are

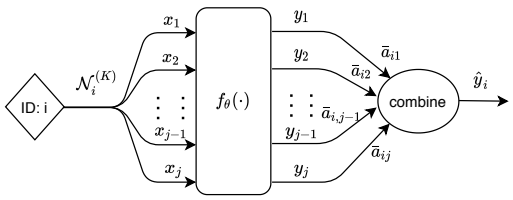


**Figure 2: APPNP ensembles the predictions of neighbors to generate final prediction in the inference phase.**

weighted by the product of the graph proximity $\bar{a}_{ji}$ and the model prediction $f_{i,h(j)}$. This finding is highly interesting and helps us to understand the reasons of the effectiveness of decoupled GCN:

(S1) Label propagation serves as a data augmentation strategy to supplement the input labeled data. In semi-supervised node classification task, the labeled data is usually of small quantity, making it insufficient to train a good neural predictor. The model would have a large variance and easily sink into over-fitting. As such, the pseudo-labeled data augmentation helps to reduce overfitting and improves model performance.

(S2) Instead of assigning a static weight to the pseudo-labeled data, decoupled GCN dynamically adjusts the weight. On the one hand, the node that is closer to the labeled source node is given a larger weight $\bar{a}_{ji}$. As analyzed in Section 4.1, this setting matches our intuition, as closer nodes usually exhibit more similarity in their properties or labels. On the other hand, if a node has a pseudo-label highly different from the prediction, the pseudo-labeled data would obtain smaller weight. This setting makes the model more robust to the structure noise, which is common in real-world graph topology. Real-world graphs may not be ideally clean and have certain noises in edges. Two connected nodes sometimes exhibit different properties and belong to different classes. For example, in a social network, connected friends may belong to different schools; in a citation network, a paper may cite the work from other fields. The pseudo-labels propagated along such noisy inter-class edges are unreasonable and may hurt the model performance. In decoupled GCN, this bad effect could be mitigated, as the contribution of these unreliable pseudo-labeled data will be reduced. Moreover, this setting endows the model the potential to mitigate over-smoothing. The noisy signal from distant labeled nodes would be attenuated heavily by the weights.

(S3) To predict the label of a node in the inference stage, as shown in Figure 2, decoupled GCN combines the predictions of the node's $K$-hop neighbors rather than basing only on the node's own prediction. For a target node $i$, decoupled GCN finds its $K$-order neighbors and combines their predictions with the trained neural network. Such an ensemble mechanism further boosts model's performance by reducing the prediction variance.

However, some weaknesses of the decoupled GCN are revealed.

(W1) Since the weights of pseudo-labeled data are dynamically adjusted based on model prediction, the initialization of the model exerts a much larger impact on the model training. An ill initialization would generate incorrect predictions in the beginning, making the weights of pseudo-labeled data diverge from the proper values. It further skews the contribution of pseudo-labeled data and the model may converge to undesirable states.

(W2) The weights of the pseudo-labeled data generated from a labeled source node are normalized to be unity, *i.e.,* $\sum_{i\in\mathcal{V}} w_{ij} = 1$. It implies that different labeled nodes are assumed to have an equal contribution to weigh the pseudo-labeled data. Such an assumption ignores the quality or importance of labeled data, which may not be ideally clean and have certain noises in practical applications. Some labels are carefully labeled by expert, whereas some labels may be contaminated by accidents or envirnmental noises. Treating them equally is not reasonble. As such, the model is vulnerable to label noises which will deteriorate the model performance.

## 4 PROPOSED METHOD: PROPAGATION THEN TRAINING ADAPATIVELY

Given the analysis of decoupled GCN, in this section, we aim to develop a better method that can foster its merits and overcome its weaknesses. We name the proposed method as *Propagation then Training Adaptively (PTA)*, which improves the LP view of decoupled GCN with an adaptive weighting strategy. Focusing on the two weaknesses of decoupled GCN, PTA makes two revisions:

(1) To make the model more robust to label noise, we remove the normalization of weights of decoupled GCN to let different labeled data exert varying impact on model training. The weight without normalization can be written as:

$$w_{ij} = \bar{a}_{ji} f_{i,h(j)}. \tag{15}$$

The accumulated weight of the pseudo-labeled data generated from a specific labeled source node can be written as:

$$S_j = \sum_{i\in V} \bar{a}_{ji} f_{i,h(j)}. \tag{16}$$

which is dynamically adjusted according to $f_{i,h(j)}$. Specifically, the importance of each labeled data $S_j$ is determined by the consistence between its label and the predicted labels of its neighbors (or multi-hop neighbors). When the labels of its neighbors are predicted to be different from $h(j)$ (*i.e.,* $f_{i,h(j)}$ is small), it implies the labeled data may be contaminated by noises. Naturally, this design reduces the contribution of this unreliable labeled data and makes the model more robust to label noise. Another advantage of removing normalization is making the model more concise and easier to implement, as the computationally expensive summation over the neighbors (or multi-hop neighbors) is avoided. Nevertheless, this design may increase model's sensitivity to the model initialization which also determines the impact of the labeled data. We address this issue in the next design.

(2) The sensitivity to model initialization is caused by the weights $f_{i,k(j)}$. However, blindly removing $f_{i,k(j)}$ is problematic as it would hurt model robustness to label noise and structure noise. To deal with this problem, we develop an adaptive weighting strategy as follows:

$$w_{ij} = \bar{a}_{ji} f_{i,h(j)}^{\gamma}, \qquad \gamma = \log(1 + e/\epsilon), \tag{17}$$

where $\gamma$ is define to control the impact of $f_{i,h(j)}$ on the weighting of the pseudo-labeled data, which will evolve with the training process. $e$ denotes the current training epoch and $\epsilon$ is a temperature hyper-parameter controlling the sensitivity of $\gamma$ to $e$. Our design is simple but rather effective. In the early stage of training, when the

immature neural network generates relatively unreliable prediction, PTA reduces the impact of model prediction on weighting pseudo-labeled data. This setting makes the model yield stable results. With the training proceeding, as the neural predictor gradually gives more accurate results, PTA enlarges its impact to make the model robust to both label noise and structure noise.

Through the two designs, we eliminate the limitations of decoupled GCN and meanwhile make full use of its advatanges. Overall, PTA optimizes the following objective function:

$$L_{PTA}(\theta) = \tilde{\sum}_{i \in V, j \in V_l} w_{ij} \, \mathrm{CE}\left(\boldsymbol{f}_i, \boldsymbol{y}_j\right), \qquad w_{ij} = \bar{a}_{ji} f_{i,h(j)}^{\gamma}. \tag{18}$$

We also give a concise matrix-wise formula as follows:

$$L_{PTA}(\theta) = -SUM\left(\boldsymbol{Y}_{soft} \otimes \boldsymbol{f}(\boldsymbol{X})^{\gamma} \otimes log\left(\boldsymbol{f}_\theta(\boldsymbol{X})\right)\right), \tag{19}$$

where $\otimes$ represents the element-wise product, $SUM(\cdot)$ represents the sum of all elements in matrix, $\boldsymbol{Y}_{soft}$ represents the soft label generated using label propagation, $\boldsymbol{f}(\boldsymbol{X})$ does not propagate gradients backward, and gradients propagate only from $log\left(\boldsymbol{f}_\theta(\boldsymbol{X})\right)$. The proof for Equation 19 can be seen in Appendix C. The complete framework of PTA is in Appendix D.

Compared the concise form of PTA with vanilla decoupled GCN (*e.g.,* APPNP), PTA also has an advantage of efficient computation. In decoupled GCN, both feature transformation and neighbor aggregation need to be conducted in each epoch. Moreover, the updating of the transformation function $\boldsymbol{f}_\theta(\boldsymbol{X})$ requires back propagation along both operations. The complexity of decoupled GCN is $\mathcal{T}(\boldsymbol{f}(\boldsymbol{X})) + \mathcal{T}(\bar{\boldsymbol{A}}\boldsymbol{H})$, where $\mathcal{T}(\boldsymbol{f}(\boldsymbol{X}))$ and $\mathcal{T}(\bar{\boldsymbol{A}}\boldsymbol{H})$ denote the complexity of the two operations, respectively. In our PTA, the two operations have been thoroughly separated, *i.e.,* graph propagation can be pre-processed and used for all epochs while each training epoch only need to consider feature transformation. The overall training algorithm of our PTA is presented in Algorithm 1. As we can see, we just need to run the label propagation to generate soft-label matrix $\boldsymbol{Y}_{soft}$, which can be considered as data pre-processing. We then train the neural predictor $\boldsymbol{f}_\theta(\boldsymbol{X})$ with soft label $\boldsymbol{Y}_{soft}$ and additional weighting $\boldsymbol{f}(\boldsymbol{X})^{\gamma}$, avoiding the expensive forward and back propagation of the neighbor aggregation in each epoch. The complexity of PTA is reduced from $\mathcal{T}(\boldsymbol{f}(\boldsymbol{X})) + \mathcal{T}(\bar{\boldsymbol{A}}\boldsymbol{H})$ to $\mathcal{T}(\boldsymbol{f}(\boldsymbol{X}))$.

It is worth mentioning that the neighborhood aggregation mechanism of GCN is non-trivial to implement in parallel [26, 29], making it the efficiency bottleneck on large graphs. In contrast, the label propagation used in PTA is well studied and much easier to implement on large graphs. As such, our PTA eases the implementation of graph learning and has great learning potentials to be deployed in large-scale industrial applications.

## 5 EXPERIMENTS

We conduct experiments on four real-world benchmark datasets to evaluate the performance of existing decoupled methods and our proposed PTA. We aim to answer the following research questions:

**RQ1:** Do three advantages of decoupled GCN that we analyzed from LP perspective indeed boost its performance?

**RQ2:** Is decoupled GCN sensitive to the initialization and the label noise? How does PTA overcome these problems?

**Algorithm 1** Propagation then Training Adaptively.

**Input:** Graph $G = (\mathcal{V}, \mathcal{E})$; Features $\boldsymbol{X}$; Observed labels $\boldsymbol{Y}$;
**Output:** Neural network predictor $\boldsymbol{y} = \boldsymbol{f}_\theta(\boldsymbol{x})$ .

```
Generate soft label matrix Ŷ_soft with label propagation:
for e = 1 to Epoch_max do
    Calculate the adaptive factor: γ = log(1 + e/ε)
    Calculate loss: L = −SUM(Y_soft ⊗ f(X)^γ ⊗ log(f_θ(X)))
    Optimize θ by minimizing L + λL_reg with gradient descent
end for
return Neural network predictor y = f_θ(x)
```

**Table 2: Statistics of the datasets.**

| Dataset | Nodes | Edges | Features | Classes |
|---|---|---|---|---|
| CITESEER | 2,110 | 3,668 | 3,703 | 6 |
| CORA_ML | 2,810 | 7,981 | 2,879 | 7 |
| PUBMED | 19,717 | 44,324 | 500 | 3 |
| MS_ACADEMIC | 18,333 | 81,894 | 6,805 | 15 |

**RQ3:** How does the proposed PTA perform as compared with state-of-the-art GCN methods?

**RQ4:** Is PTA more efficient than decoupled GCN?

### 5.1 Experimental Setup

We take APPNP as the representative model of decoupled GCN for experiments, since APPNP performs similarly with DAGNN while they are both superior over SGCN. To reduce the experiment workload and keep the comparison fair, we closely follow the settings of the APPNP work [16], including the experimental datasets and various implementation details.

*Datasets.* Following the APPNP work [16], we also use four node-classification benchmark datasets for evaluation, including CITESEER [27], CORA_ML [21], PUBMED [23] and MICROSOFT ACADEMIC [28]. CITESEER, CORA_ML, and PUBMED are citation networks, where each node represents a paper and an edge indicates a citation relationship. MICROSOFT ACADEMIC is a co-authorship network, where nodes and edges represent authors and co-author relationship, respectively. The dataset statistics is summarized in Table 2. Also, we follow [16] and split each dataset into training set, early-stopping set, and test set, where each class has 20 labeled nodes in the training set.

*Compared methods.* The main competing method is APPNP, which has shown to outperform several graph-based models, including Network of GCNs (N-GCN) [1], graph attention network (GAT) [30], jumping knowledge networks with concatenation (JK) [37], etc. As the comparison is done on the same datasets under the same evaluation protocol, we do not further compare with these methods. In addition to APPNP, we further compare with two relevant and competitive decoupled methods: DAGNN [19] and SGCN [32], and two baseline models MLP, GCN [15]. Further two variants of PTA are tested:

- **PTS**: propagation then training statically, where we give a graph-based static weighting $a_{ji}$ to pseudo-labeled data. PTS can be considered as simple version of APPNP and PTA,

**Table 3: Characteristics of APPNP and its variants.**

| Method | Pseudo-labels? | Graph-based weighting? | Model-based weighting? | Ensemble? |
|---|---|---|---|---|
| MLP | × | × | × | × |
| APPNP-noa-nof | X | × | × | X |
| PTS | X | X | × | X |
| APPNP-noe | X | X | X | × |
| APPNP | X | X | X | X |

**Table 4: Performance of APPNP and its variants.**

| Method | CITESEER | CORA_ML | PUBMED | MS_ACA |
|---|---|---|---|---|
| MLP | 63.98 ± 0.44 | 68.42 ± 0.34 | 69.47 ± 0.47 | 89.69 ± 0.10 |
| APPNP-noa-nof | 72.71 ± 0.55 | 78.51 ± 0.46 | 77.18 ± 0.53 | 90.18 ± 0.23 |
| PTS | 75.58 ± 0.25 | 85.02 ± 0.24 | 79.67 ± 0.28 | 92.76 ± 0.10 |
| APPNP-noe | 70.98 ± 0.34 | 77.74 ± 0.27 | 74.80 ± 0.43 | 89.85 ± 0.09 |
| APPNP | 75.48 ± 0.29 | 85.07 ± 0.25 | 79.61 ± 0.33 | 93.31 ± 0.08 |

where model-based weighting is removed. More details about PTS can refer to section 3.1.

- **PTD**: propagation then training dynamically. PTD can be considered as an intermediate model between APPNP and PTA, where each pseudo-labeled data is weighed by $a_{ji}f_{i,h(j)}$. Comparing with APPNP, weighting normalization has been removed; Comparing with PTA, PTD does not use the adaptive factor.

*Implementation details.* In our experiments, we take APPNP as the representative model of decoupled GCN for experiments. For fair comparison, all the configures of PTD, PTS, PTA are the same as APPNP including layers, hidden units, regularization, early stopping, initialization and optimizer. Also, we use the same propagation scheme as APPNP. That is, we use PageRank to propagate labels with the same hyper-parameters. The additional parameter $\varepsilon$ in Equation (17) is set as 100 across all datasets. We run each experiment 100 times on multiple random splits and initialization. More details of the setting of our PTA are presented in Appendix E. For the compared methods, we refer to their results or settings that are reported in their papers. We will share our source code when the paper gets published.

## 5.2 Empirical Analyses of APPNP (RQ1)

In this subsection, we take APPNP as a representative method to answer the RQ1. As discussed in section 3.3, we attribute the effectiveness of decoupled GCN to three different aspects: 1) data augmentation through label propagation, 2) dynamic weighting, and 3) Ensembles multiply predictors. To show the impact of these aspects, we conduct ablation studies and compare APPNP with its four variants: 1) MLP, where both pseudo-labels and ensemble are removed; 2) APPNP-noa-nof, where weighting of pseudo-labels ($w_{ij}$) is removed; 3) PTS, where dynamic weighting $f_{i,h(j)}$ is removed. 4) APPNP-noe, where the ensemble is removed; The characteristics of these compared methods are presented in Table 3.

**Effect of label propagation.** From Table 4, we can find all the methods with label propagation outperform MLP by a large margin. Specifically, the average accuracy improvement of APPNP (or even if its non-ensemble version APPNP-noe) over MLP on four datasets is 7.40% (or 5.45%), which are rather significant. Moreover, to our surprise, the simple model PTS achieves impressive performance. Although its performance is relatively inferior to APPNP, the values are pretty close. This result validates the major reason of the effectiveness of APPNP is data augmentation with label propagation.

**Effect of weighting.** On the one hand, by comparing PTS with APPNP-noa-nof, we can conclude that weighing the pseudo-labeled data with graph proximity $\bar{a}_{ji}$ is highly useful. On the other hand, we observe APPNP relatively performs better than PTS as it introduces an additional dynamic weighting $f_{i,h(j)}$, which can mitigate adverse effects of structure noise.

To gain more insight in the effect of dynamic weighting, we conduct an additional experiment to explore the robustness of these methods to the structure noise. Here we first define the *structure noise rates* as the percent of the "noisy" edges in the graph, where the "noisy" edges denote the edges whose connected nodes belong to different classes. We randomly transfer some good (noisy) edges into noisy (good) edges to generate simulated graphs with different structure noise rates. The performance of APPNP, PTS, GCN, MLP are presented in Figure 3. For APPNP and PTS, here we choose 2-layer graph propagation for fair comparison with GCN. The black dash line denotes the structure noise rates of the original graph.

As we can see from Figure 3, when the graph is relative clean (less than 5%), all the graph-based methods perform pretty well. But it is impractical due to the collected graph are always not so clean. As the structure noise increases, the performance of original GCN drops quickly, which vividly validate the superiority of decoupled GCN over vanilla GCN. Also, we observe the margin achieved by APPNP over PTS become larger with the noise increasing. This experimental result is consistent with our analysis presented in section 3.3 that model-based dynamic weighting indeed improve model's robustness to structure noise.

More interestingly, when the graph structure noise is large enough (over 50% in the four data-sets), all GNN-based models (GCN, APPNP, PTS) perform even worse than MLP. This phenomenon validates the basic assumption of GCN-based methods is *local homogeneity* [38], *i.e.,* connected nodes in the graph tend to share similar properties and same labels. When the assumption is not hold in the graph, this kind of methods may not be applicable.

**Effect of ensemble.** We also observe that APPNP consistently outperforms APPNP-noe. This result validates that ensemble is anther important factor of the effectiveness of decoupled GCN.

## 5.3 Study of Robustness (RQ2)

In this subsection, we explore how PTA compares with APPNP in terms of robustness to the initialization and label noise. The robustness to structure noise can refer to figure 6 in Appendix F.

**Robustness to initialization.** Figure 4 shows the accuracy distribution of each model is. The smaller boxes and the less outlines suggest the model is more robust to the initialization. From the Figure 4 and Table 4, We observe that: 1) APPNP, which introduces the dynamic weighting on pseudo-labeled data, is more sensitive to the initialization than PTS. 2) PTA, which adopts the adaptive weighting strategy, is more stable than APPNP and PTD. 3) PTD, where the normalization of weights is removed, is highly unstable.

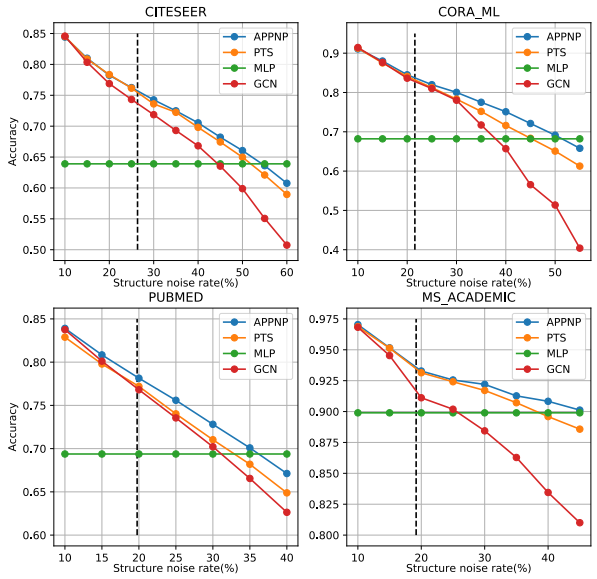


**Figure 3: The model robustness to graph structure noise of different models. The black dashed line indicates the structure noise rate of the original graph.**

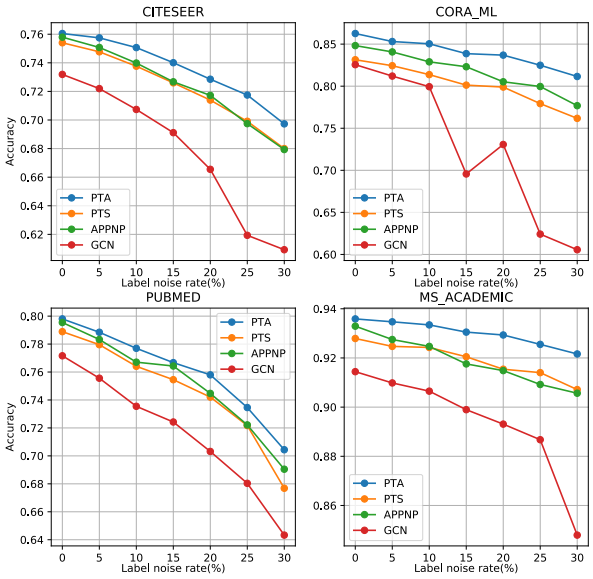


**Figure 5: The model robustness to the label noise.**

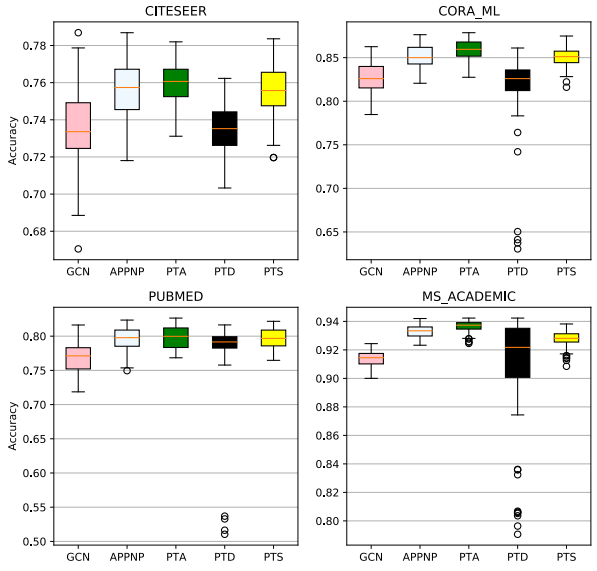


**Figure 4: Accuracy distributions of the compared models.**

These results validate the necessary and effectiveness of introducing the adaptive weighting strategy. It can reduce the impact of model initialization and foster model's robustness to the noises.

**Robustness to the label noise.** We also conduct an interesting simulated experiments to explore the robustness of the models to the label noise. That is, we randomly transfer a certain number of instances in each class and change their labels to others ("noisy" labels). We then run different methods on these simulated graph with different ratios of "noisy" labels. the result is presented in Figure 5. We observe that: 1) Even there is no label noise, PTA still outperforms others. This result suggests that the equal-contribution assumption is invalid. Different labeled data usually has different quality and importance on learning a neural network predictor. 2) The margins achieved by PTA over APPNP and PTA become larger with the noise increasing. This result is consistent with our analysis presented in section 4 that removing normalization of weights indeed boosts model's robustness to the label noise.

## 5.4 Performance Comparison with State-of-the-Arts (RQ3)

Table 5 presents the performance of compared methods in terms of accuracy. The boldface font denotes the winner in that column. For the important baseline APPNP, we present both the results that are reported in the original paper with mark '#', and our reproduced results. They may have slight difference due to random initialization and random data splits. To ensure the statistical robustness of our experimental setup, we calculate confidence intervals via bootstrapping and report the p-values of a paired t-test between PTA and APPNP. From the table, we have the following observations:

Overall, PTA outperforms all compared methods on all datasets. This result validates our proposed PTA benefits to train a better neural network predictor. Specifically, comparing with APPNP, the best baseline in general, the improvements of PTA over APPNP is statistically significant with paired t-test at $p < 0.05$ on all datasets.

## 5.5 Efficiency Comparison (RQ4)

In this section, we empirically compare the efficiency of PTA and APPNP. Table 6 shows the running time of PTA and APPNP in each epoch, while Table 7 shows the total time cost on training the

**Table 5: Accuracy of our PTA comparing with state-of-the-art methods. The $p$-value in the last row is obtained via a paired $t$-test between PTA and APPNP.**

| Method | CITESEER | CORA_ML | PUBMED | MS_ACA |
|---|---|---|---|---|
| MLP | 63.98 ± 0.44 | 68.42 ± 0.34 | 69.47 ± 0.47 | 89.69 ± 0.10 |
| GCN | 73.62 ± 0.39 | 82.70 ± 0.39 | 76.84 ± 0.44 | 91.39 ± 0.10 |
| SGCN | 75.57 ± 0.28 | 75.97 ± 0.72 | 71.24 ± 0.86 | 91.03 ± 0.16 |
| APPNP # | 75.73 ± 0.30 | 85.09 ± 0.25 | 79.73 ± 0.31 | 93.27 ± 0.08 |
| DAGNN | 74.53 ± 0.38 | 85.75 ± 0.23 | 79.59 ± 0.37 | 92.29 ± 0.07 |
| APPNP | 75.48 ± 0.29 | 85.07 ± 0.25 | 79.61 ± 0.33 | 93.31 ± 0.08 |
| PTA | **75.98 0.24** | **85.90 0.21** | **79.89 0.31** | **93.64 0.08** |
| p-value | $5.56 \times 10^{-4}$ | $1.81 \times 10^{-9}$ | $1.09 \times 10^{-2}$ | $1.57 \times 10^{-8}$ |

**Table 6: The training time per epoch of PTA and APPNP.**

| Method | CITESEER | CORA_ML | PUBMED | MS_ACA |
|---|---|---|---|---|
| APPNP | 34.73ms | 28.60ms | 34.98ms | 30.51ms |
| PTA | **3.33*ms*** | **3.35*ms*** | **3.27*ms*** | **3.33*ms*** |

**Table 7: The total training time of PTA and APPNP.**

| Method | CITESEER | CORA_ML | PUBMED | MS_ACA |
|---|---|---|---|---|
| APPNP | 52.75s | 75.30s | 49.39s | 134.23s |
| PTA | 10.14*s* | 11.95*s* | 10.59*s* | 17.12s |
| PTA(F) | **1.19*s*** | **1.25*s*** | **1.40*s*** | **3.92*s*** |

**Table 8: The accuracy of PTA(F).**

| Method | CITESEER | CORA_ML | PUBMED | MS_ACA |
|---|---|---|---|---|
| APPNP | 75.48 ± 0.29 | 85.07 ± 0.25 | 79.61 ± 0.33 | 93.31 ± 0.08 |
| PTA(F) | 75.51 ± 0.24 | 85.73 ± 0.22 | 79.45 ± 0.40 | 93.62 ± 0.08 |
| PTA | **75.98 0.24** | **85.90 0.21** | **79.89 0.31** | **93.64 0.08** |

two models. Note that estimating performance of PTA on the early-stopping set is time-consuming, we further design a fast mode of PTA (PTA(F)), which directly use $f_\theta(x)$ instead of ensemble results for early-stopping estimation. The performance of PTA(F) comparing with PTA and APPNP is presented in Table 8. From the tables, We can conclude that PTA is much faster than APPNP: on average, about 9.7 times acceleration per epoch and 5.7 times acceleration for totally running time. When we use fast mode (PTA(F)), although its performance would decay slightly (-0.28% on average), it still outperforms APPNP and achieves impressive speed-up (about 43 times over APPNP and 7.5 times over PTA).

## 6 RELATED WORK

Inspired by the success of the *convolutional neural networks* (CNN) in computer vision [10], CNN has been generalized to graph-structured data with a so-called *graph convolutional neural network* (GCN). There are two lines to understand GCN: From a spectral perspective, the convolutions on the graph can be understood as a filter to remove the noise from graph signals. The first work on this line was presented by Bruna *et al.* [2]. Defferrard *et al.* [6] and Kipf *et al.* [15] further propose to simplify graph convolutions with Chebyshev polynomial to avoid expensive computation of graph Laplacian eigenvectors. Afterwards, some researchers put forward their understanding of GCN in spatial domain [36]. From the spatial perspective, the convolution in GCN can be analogized with "patch operator" which refines the representation of each node by combining the information from its neighbors. This simple and intuitive thinking inspires much work on spatial GCN [4, 9, 30, 36], which exhibits attractive efficiency and flexibility, and gradually becomes the mainstream. For example, Velickovic *et al.*[30] leveraged attention strategy in GCN, which specifies different weights to different neighbors; Hamilton [9] *et al.*[36] proposed to sample a part of neighbors for model training to make the GCN scale up to large-scale graph; Xu *et al.*analyzed the expressive power of GCN to capture different graph structures, and further proposed *Graph Isomorphism Network*; Li *et al.*[18] validate the GCN is a special form of Laplacian smoothing and show exsting methods may suffer from over-smoothing issue; Chen *et al.*[4] further explored over-smoothing issue of GCN and propose two strategies MADGap and AdaEdge to address it. Here we just list most related work. There are many other graph neural models. We refer the readers to excellent surveys and monographs for more details [3, 33].

Note that *feature transformation* and *neighborhood aggregation* are two important operations in a spatial GCN model. In the original GCN [15] and many follow-up models [20, 25, 31], the two operations are coupled, where each operation is companied with the other in a graph convolution layer. In fact, some recent works find such a coupling design is unnecessary and even troublesome. Klicpera *et al.* [16] and Liu *et al.* [19] claim that decoupling the two operations permits more deep propagation without leading to over-smoothing; Wu *et al.* [32] and He *et al.* [11] empirically validate the simplified decoupled GCN outperforms the vanilla one in terms of both accuracy and efficiency. Despite decoupled GCN attracts increasing attention and has become the last paradigm of GCN, to our best knowledge, none of work has provided deep analysis of working mechanisms, advantages and limitations about it.

Lastly, both label propagation (LP) [34] and GCN follow the information propagation scheme, which implies they have the same internal foundation. However, their relations have not been investigated. The most relevant work that jointly considers both methods is [12], which trains a base predictor on the labeled data and then corrects it by information propagation on the graph. The angle of this work differs from ours significantly. In this work, we prove that decoupled GCN is identical to the *Propagation then Training*, which to our knowledge is the first work that reveals the essential relation between LP and GCN.

## 7 CONCLUSION

In this work, we conduct thorough theoretical analyses on decoupled GCN and prove its training stage is essentially equivalent to performing *Propagation then Training*. This novel view of label propagation reveals the reasons of the effectiveness of decoupled GCN: 1) data augmentation through label propagation; 2) structure- and model- aware weighting for the pseudo-labeled data; and 3) combining the predictions of neighbors. In addition to the advantages, we also identify two limitations of decoupled GCN — sensitive to model initialization and to label noise. Based on these insight, we further propose a new method *Propagation then Training adaptive* (PTA), which posters the advantages of decoupled GCN and

overcomes its weaknesses by introducing an adaptive weighting strategy. Empirical studies on four node classification datasets validate the superiority of the proposed PTA over decoupled GCNs in all robustness, accuracy, and efficiency.

We believe the insights brought by the label propagation view are inspiration for future research and application of GCN. Here we point out three directions. First, our analyses focus on the semi-supervised node-classification setting. How to extend to other tasks like link prediction and graph classification is interesting and valuable to explore. Second, this work provides a new view of GCN. It will be useful to explore existing methods from this perspective and analyze their pros and cons, which are instructive to develop better models. Third, our proposed PTA uses a relatively simple weighting strategies. More sophisticated weighting strategy can be explored, such as learning from side information, graph global topology, validation data, or employing adversarial learning for better robustness.

## ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (U19A2079, U1936210) and the National Key Research and Development Program of China (2020AAA0106000).

## REFERENCES

[1] Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. 2019. N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification. In *the Conference on Uncertainty in Artificial Intelligence, UAI, 2019.*

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *the International Conference on Learning Representations, ICLR, 2014.*

[3] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering, TKDE* (2018).

[4] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. *AAAI Conference on Artificial Intelligence, AAAI* (2020).

[5] Weijian Chen, Yulong Gu, Zhaochun Ren, Xiangnan He, Hongtao Xie, Tong Guo, Dawei Yin, and Yongdong Zhang. 2019. Semi-supervised User Profiling with Heterogeneous Graph Attention Networks. In *the International Joint Conference on Artificial Intelligence, IJCAI 2019.*

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems, NIPS, 2016.*

[7] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Benjamin Chamberlain, Michael Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. In *International Conference on Machine Learning, ICML 2020.*

[8] Vikas K. Garg, Stefanie Jegelka, and Tommi S. Jaakkola. 2020. Generalization and Representational Limits of Graph Neural Networks. In *International Conference on Machine Learning, ICML, 2020.*

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems, NIPS, 2017.*

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Conference on Computer Vision and Pattern Recognition,CVPR, 2016.*

[11] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020.*

[12] 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 *the International Conference on Learning Representations, ICLR 2021.*

[13] Di Jin, Ziyang Liu, Weihao Li, Dongxiao He, and Weixiong Zhang. 2019. Graph convolutional networks meet markov random fields: Semi-supervised community detection in attribute networks. In *the AAAI Conference on Artificial Intelligence, AAAI, 2019.*

[14] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations, ICLR,2015.*

[15] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations, ICLR, 2017.*

[16] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations, ICLR, 2019.*

[17] Boris Knyazev, Graham W. Taylor, and Mohamed R. Amer. 2019. Understanding Attention and Generalization in Graph Neural Networks. In *Conference on Neural Information Processing Systems, NIPS, 2019.*

[18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI Conference on Artificial Intelligence, AAAI, 2018.*

[19] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards Deeper Graph Neural Networks. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD, 2020.*

[20] Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and Wenwu Zhu. 2019. Disentangled Graph Convolutional Networks. *International Conference on Machine Learning, ICML* (2019).

[21] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* (2000).

[22] Kevin P Murphy. 2012. *Machine learning: a probabilistic perspective.* MIT press.

[23] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. 2012. Query-driven active surveying for collective classification. In *International Workshop on Mining and Learning with Graphs, 2012.*

[24] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report. Stanford InfoLab.

[25] Aravind Sankar, Junting Wang, Adit Krishnan, and Hari Sundaram. 2020. Beyond Localized Graph Neural Networks: An Attributed Motif Regularization Framework. In *the International Conference on Data Mining, ICDM 2020.*

[26] Simone Scardapane, Indro Spinelli, and Paolo Di Lorenzo. 2021. Distributed Training of Graph Convolutional Networks. *IEEE Trans. Signal Inf. Process. over Networks* (2021).

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

[28] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. In *Relational Representation Learning Workshop, R2L, 2018.*

[29] Chao Tian, Lingxiao Ma, Zhi Yang, and Yafei Dai. 2020. PCGCN: Partition-Centric Processing for Accelerating Graph Convolutional Network. In *2020 IEEE International Parallel and Distributed Processing Symposium IPDPS, 2020.*

[30] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations, ICLR, 2018.*

[31] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR, 2019.*

[32] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning, ICML, 2019.*

[33] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[34] Zhu Xiaojin and Ghahramani Zoubin. 2002. Learning from labeled and unlabeled data with label propagation. *Technical Report, Carnegie Mellon University* (2002).

[35] Yiqing Xie, Sha Li, Carl Yang, Raymond Chi-Wing Wong, and Jiawei Han. 2020. When Do GNNs Work: Understanding and Improving Neighborhood Aggregation. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020.*

[36] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations, ICLR, 2019.*

[37] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *International Conference on Machine Learning, ICML, 2018.*

[38] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *International Conference on Machine Learning, ICML, 2016.*

[39] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph convolutional networks for text classification. In *the AAAI Conference on Artificial Intelligence, AAAI, 2019.*

[40] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware Graph Neural Networks. In *the International Conference on Machine Learning, ICML 2019.*

[41] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. In *Neural Information Processing Systems, NeurIPS, 2018.*

[42] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. In *Conference on Artificial Intelligence, AAAI, 2018.*
[43] 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, NIPS, 2004.*
[44] Hongmin Zhu, Fuli Feng, Xiangnan He, Xiang Wang, Yan Li, Kai Zheng, and Yongdong Zhang. 2020. Bilinear Graph Neural Network with Neighbor Interactions. In *the International Joint Conference on Artificial Intelligence, IJCAI 2020.*
[45] Chenyi Zhuang and Qiang Ma. 2018. Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018.*

# A APPNP TO THE ARCHITECTURE OF DECOUPLED GCN

The general architecture of *decoupled GCN* can be written:

$$\hat{\boldsymbol{Y}} = \text{softmax}\ \ \bar{\boldsymbol{A}}\boldsymbol{f}_\theta(\boldsymbol{X})\ \ , \tag{20}$$

and the formulation of APPNP is:

$$\begin{aligned}
&\boldsymbol{H}^{(0)} = \boldsymbol{f}_\theta(\boldsymbol{X})\\
&\boldsymbol{H}^{(k)} = (1-\alpha)\hat{\boldsymbol{A}}\boldsymbol{H}^{(k-1)} + \alpha\boldsymbol{H}^{(0)}, \quad k = 1, 2, \ldots K-1\\
&\hat{\boldsymbol{Y}} = \text{softmax}\ \ \boldsymbol{H}^{(K)}\ \ .
\end{aligned} \tag{21}$$

Now we prove that APPNP can be subsumed into the architecture of decoupled GCN, with $\bar{\boldsymbol{A}} = (1-\alpha)^K\hat{\boldsymbol{A}}^K + \alpha\sum_{k=0}^{K-1}(1-\alpha)^k\hat{\boldsymbol{A}}^k$.

Proof. We prove it by mathematical induction.

*Base case:*

When $K = 1$, we have $\hat{\boldsymbol{Y}} = \text{softmax}\ \ [(1-\alpha)\hat{\boldsymbol{A}} + \alpha\boldsymbol{I}]\boldsymbol{f}_\theta(\boldsymbol{X})$ and $\bar{\boldsymbol{A}}^{(1)} = (1-\alpha)\hat{\boldsymbol{A}} + \alpha\boldsymbol{I}$, which satisfies $\bar{\boldsymbol{A}}^{(1)} = (1-\alpha)^K\hat{\boldsymbol{A}}^K + \alpha\sum_{k=0}^{K-1}(1-\alpha)^k\hat{\boldsymbol{A}}^k$.

*Inductive step:*

Assume the induction hypothesis that for a particular $K \geq 1$ the equations $\hat{\boldsymbol{Y}} = \text{softmax}\ \ \bar{\boldsymbol{A}}^{(K)}\boldsymbol{H}^{(0)}$, $\bar{\boldsymbol{A}}^{(K)} = (1-\alpha)^K\hat{\boldsymbol{A}}^K + \alpha\sum_{k=0}^{K-1}(1-\alpha)^k\hat{\boldsymbol{A}}^k$ hold. Then we have:

$$\begin{aligned}
\hat{\boldsymbol{Y}} &= \text{softmax}\ \ (1-\alpha)\hat{\boldsymbol{A}}\boldsymbol{H}^{(K)} + \alpha\boldsymbol{H}^{(0)}\\
&= \text{softmax}\ \ (1-\alpha)\hat{\boldsymbol{A}}\bar{\boldsymbol{A}}^{(K)}\boldsymbol{H}^{(0)} + \alpha\boldsymbol{H}^{(0)}\\
&= \text{softmax}\ \ [(1-\alpha)\hat{\boldsymbol{A}}\bar{\boldsymbol{A}}^{(K)} + \alpha\boldsymbol{I}]\boldsymbol{H}^{(0)}\\
&= \text{softmax}\ \ [(1-\alpha)^{(K+1)}\hat{\boldsymbol{A}}^{(K+1)} + \alpha\sum_{k=0}^{K}(1-\alpha)^k\hat{\boldsymbol{A}}^k]\boldsymbol{f}_\theta(\boldsymbol{X})\ \ ,
\end{aligned} \tag{22}$$

which satisfies $\bar{\boldsymbol{A}}^{(K+1)} = (1-\alpha)^{(K+1)}\hat{\boldsymbol{A}}^{(K+1)} + \alpha\sum_{k=0}^{K}(1-\alpha)^k\hat{\boldsymbol{A}}^k$.

Therefore, APPNP can be subsumed into the form of decoupled GCN, with $\bar{\boldsymbol{A}} = (1-\alpha)^K\hat{\boldsymbol{A}}^K + \alpha\sum_{k=0}^{K-1}(1-\alpha)^k\hat{\boldsymbol{A}}^k$.

# B SOFTMAX FUNCTION

The original formulation of *decoupled GCN* can be written as:

$$\hat{Y} = \text{softmax}\ \ \bar{A}\boldsymbol{f}_\theta(X)\ \ . \tag{23}$$

The role of the outer softmax$(\cdot)$ function is to normalize the output into a probability distribution, which is reasonable for optimizing the model with the cross entropy loss.

In our concise formulation of decoupled GCN, the softmax function has been integrated into feature transformation function as follows:

$$\begin{aligned}
\boldsymbol{f}_\theta(\boldsymbol{X}) &= \text{softmax}(\boldsymbol{f}_\theta(\boldsymbol{X}))\\
\hat{\boldsymbol{Y}} &= \bar{\boldsymbol{A}}\boldsymbol{f}_\theta(\boldsymbol{X}).
\end{aligned} \tag{24}$$

The major concern is whether its output is reasonable for cross entropy loss. We now show that although the normalization of the output from our concise model may not be hold, it is equivalent to optimize the following objective function, where we give normalized prediction for cross entropy loss.

For arbitrary node $i$ in the training set, the loss function is:

$$\begin{aligned}
l(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i) &= -\sum_{k\in C} y_{ik}log\,(\hat{y}_{ik})\\
&= -\sum_{k\in C} y_{ik}log\frac{\sum_{j\in V}\bar{a}_{ij}f_{jk}}{\sum_{j\in V}\bar{a}_{ij}}\\
&= -\sum_{k\in C} y_{ik}log\frac{\sum_{j\in V}\bar{a}_{ij}f_{jk}}{\sum_{q\in V}\bar{a}_{iq}} - \sum_{k\in C} y_{ik}log\sum_{j\in V}\bar{a}_{ij}\\
&= -\sum_{k\in C} y_{ik}log\sum_{j\in V} g_{ij}f_{jk} - log\sum_{j\in V}\bar{a}_{ij},
\end{aligned} \tag{25}$$

where $g_{ij} = \frac{\bar{a}_{ij}}{\sum_{q\in V}\bar{a}_{iq}}$, satisfying $\sum_{j\in V} g_{ij} = 1$. Thus, we have the normalized prediction $\sum_{k\in C}\sum_{j\in V} p_{ij}f_{jk} = 1$, which meets the constraints. We can find the first term of the last line in Equation (25) is a cross entropy loss between the normalized prediction and labels, and the second term is a constant.

# C CONCISE MATRIX-WISE LOSS FUNCTION

Overall, PTA optimizes the following objective function:

$$L_{PTA}(\theta) = \sum_{i\in V, j\in V_l} w_{ij}\,\text{CE}\ \ \boldsymbol{f}_i, \boldsymbol{y}_j\ \ , \qquad w_{ij} = \bar{a}_{ji}f^{Y}_{i,h(j)}. \tag{26}$$

The equivalent matrix-wise formulation is concise as follows:

$$L_{PTA}(\theta) = -SUM\ \ \boldsymbol{Y}_{soft} \otimes \boldsymbol{f}(\boldsymbol{X})^{Y}_{detach} \otimes log\,(\boldsymbol{f}(\boldsymbol{X}))\ \ . \tag{27}$$

where $\otimes$ represents the element-wise product, the subscript "detach" denotes no gradient will be backward-propagated along this term, and the $SUM(\cdot)$ represents the sum of all elements on the matrix.

Proof. First, let's review the properties of $\bar{\boldsymbol{A}}$ and $\boldsymbol{y}_i$.

(1) Note that $\bar{\boldsymbol{A}}$ is calculated from an adjacency matrix $\hat{\boldsymbol{A}}$ with a specific propagation strategy (*i.e.,* $\bar{\boldsymbol{A}} = \sum_k \beta_k\hat{\boldsymbol{A}}^k$). In an undirected graph, since $\boldsymbol{A}$ is a symmetric matrix, we can conclude $\hat{\boldsymbol{A}}$ is a symmetric matrix and $\bar{\boldsymbol{A}}$ is symmetric too, *i.e.,* $\bar{a}_{ij} = \bar{a}_{ij}$.
(2) $\boldsymbol{y}_j$ is a one-hot vector, where $h(j)$-th element of $\boldsymbol{y}_j$ is 1, *i.e.,* $y_{j,h(j)} = 1$.

We then have:

$$
\begin{aligned}
L_{PTA}(\theta) &= \sum_{i\in V, j\in V_l} w_{ij}\,\mathrm{CE}\left(\boldsymbol{f}_i, \boldsymbol{y}_j\right) \\
&= -\sum_{i\in V}\sum_{j\in V_l} \bar{a}_{ji} f_{i,h(j)}^{\gamma} \sum_{k\in C} y_{jk} log\left(f_{ik}\right) \\
&= -\sum_{i\in V}\sum_{j\in V_l} \bar{a}_{ji} f_{i,h(j)}^{\gamma} y_{j,h(j)} log\ \ f_{i,h(j)} \\
&= -\sum_{i\in V, j\in V_l} \bar{a}_{ji} \sum_{k\in C} y_{jk} f_{ik}^{\gamma} log\left(f_{ik}\right) \\
&= -\sum_{i\in V, K\in C} \left(\sum_{j\in V_l} \bar{a}_{ji} y_{jk}\right) f_{ik}^{\gamma} log\left(f_{ik}\right) \\
&= -\sum_{i\in V, K\in C} \left(\sum_{j\in V_l} \bar{a}_{ij} y_{jk}\right) f_{ik}^{\gamma} log\left(f_{ik}\right) \\
&= -SUM\ \left(Y_{soft} \otimes \boldsymbol{f}(\boldsymbol{X})_{detach}^{\gamma} \otimes log\left(\boldsymbol{f}(\boldsymbol{X})\right)\right)\ ,
\end{aligned}
\tag{28}
$$

where $Y_{soft}$ represents the soft label matrix generated by label propagation.

## D FRAMEWORK OF PTA

The framework of PTA consists of three parts: data pre-processing, training and inference.

*Data pre-processing.* We first calculate the soft label matrix $Y_{soft}$ by label propagation. There are various propagation scheme can be chosen. In this paper, we adopt Personalized PageRank for experiments, which can be written as follow:

$$
\begin{aligned}
Y^{(0)} &= Y_{training} \\
Y^{(k)} &= (1-\alpha)\hat{A}Y^{(k-1)} + \alpha Y^{(0)} \\
\boldsymbol{y}_i^{(k)} &= \boldsymbol{y}_i, \forall i \in \mathcal{V}_l \\
Y_{soft} &= Y^{(K)}.
\end{aligned}
\tag{29}
$$

$Y_{soft}$ will be used in the training stage.

*Training.* In this step, we train the neural network predictor $f_\theta(\cdot)$ by optimizing the following objective function:

$$
L(\theta) = -SUM\ \left(Y_{soft} \otimes \boldsymbol{f}(\boldsymbol{X})_{detach}^{\gamma} \otimes log\left(\boldsymbol{f}(\boldsymbol{X})\right)\right)\ . \tag{30}
$$

where $\gamma = \log(1 + e/\epsilon)$ for PTA, $\gamma = 0$ PTS and $\gamma = 1$ for PTD. Note that we usually use an additional regularization term to avoid over-fitting.

*Inference.* After training $f_\theta(\cdot)$, following APPNP, ensemble has been adopted for final prediction. The formulation is:

$$
\begin{aligned}
\boldsymbol{H}^{(0)} &= f_\theta(\boldsymbol{X}) \\
\boldsymbol{H}^{(k)} &= (1-\alpha)\hat{A}\boldsymbol{H}^{(k-1)} + \alpha\boldsymbol{H}^{(0)}, \quad k = 1, 2, \ldots K-1 \\
\hat{Y} &= \boldsymbol{H}^{(K)}.
\end{aligned}
\tag{31}
$$

## E EXPERIMANTAL DETAILS

The four datasets used in this paper are downloaded from the official implementation of APPNP [16] in Github. We also follow the data split of APPNP. Moreover, we follow the same estimation method for accuracy as APPNP, *i.e.,* the average performance across 100 different random initializations and uncertainties showing the 95% confidence level calculated by bootstrapping. All experiments are conducted on a a server with 2 Intel E5-2620 CPUs, 8 2080Ti GPUS and 512G RAM.

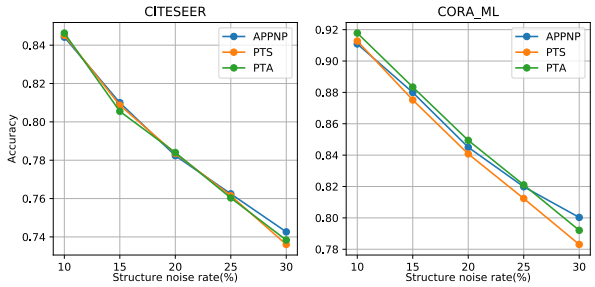


**Figure 6: The robustness of different methods to graph structure noise.**

*Hyper parameters.* For fairness, we use the same neural network model scale as the baseline models: two-layer neural network with 64 hidden units. We also use the same $K$ and $\alpha$ as APPNP, *i.e.,* $K = 10, \alpha = 0.1$ for three citation graphs, and $K = 10, \alpha = 0.2$ for co-authorship graph. The overall loss function is: $LOSS = \lambda_1 L_1 + \lambda_2 L_2$, where $L_1$ represents loss in Equation 19, and $L_2$ represents regularization on the weights of the first neural network layer. We fix $\lambda_2 = 0.005$, and find the best $\lambda_1 = 0.05$. We use the Adam optimizer with a learning rate of $lr = 0.1$ [14], The dropout rate for neural model is 0.0. The addition parameter $\varepsilon$ in Equation( 17) is set as 100 for all datasets.

## F ROBUSTNESS COMPARISON TO THE STRUCTURE NOISE

Figure 6 shows the structure noise of our PTA comparing with APPNP and PTS. As the performance of these methods are so close, here we just report the noise rate from 10% to 30% to amplify the difference. Also, the noise rate of the real-world graph is always on this region. Generally speaking, we can find the performance of PTA and APPNP are better than PTS. Comparing PTA with APPNP, their performance are in the same level. But PTA performs slighter worse than APPNP when the graph has high structure noise rate (*e.g.,* rate=30%).