Meta Propagation Networks for Graph Few-shot Semi-supervised Learning

Kaize Ding[†], Jianling Wang[‡], James Caverlee[‡], and Huan Liu[†]

[†]Arizona State University
[‡]Texas A&M University
[†]{kaize.ding, huan.liu}@asu.edu
[‡]{jlwang, caverlee}@tamu.edu

Abstract

Inspired by the extensive success of deep learning, graph neural networks (GNNs) have been proposed to learn expressive node representations and demonstrated promising performance in various graph learning tasks. However, existing endeavors predominately focus on the conventional semi-supervised setting where relatively abundant gold-labeled nodes are provided. While it is often impractical due to the fact that data labeling is unbearably laborious and requires intensive domain knowledge, especially when considering the heterogeneity of graph-structured data. Under the few-shot semi-supervised setting, the performance of most of the existing GNNs is inevitably undermined by the overfitting and oversmoothing issues, largely owing to the shortage of labeled data. In this paper, we propose a decoupled network architecture equipped with a novel meta-learning algorithm to solve this problem. In essence, our framework Meta-PN infers high-quality pseudo labels on unlabeled nodes via a meta-learned label propagation strategy, which effectively augments the scarce labeled data while enabling large receptive fields during training. Extensive experiments demonstrate that our approach offers easy and substantial performance gains compared to existing techniques on various benchmark datasets.

1 Introduction

Graphs serve as a common language for modeling a plethora of structured and relational systems, ranging from social networks (Zafarani, Abbasi, and Liu 2014) to citation networks (Namata et al. 2012), from knowledge graphs (Wang et al. 2017) to molecular graphs (Klicpera, Groß, and Günnemann 2019). To ingest the rich information encoded in graph-structured data, it is of paramount importance to learn expressive node representations by modeling the information from both node attributes and graph topology. Among numerous endeavors in the graph machine learning (Graph ML) community, graph neural networks (GNNs) have received significant attention due to their effectiveness and scalability (Kipf and Welling 2017; Veličković et al. 2018; Hamilton, Ying, and Leskovec 2017).

In general, most of the prevailing GNNs adopt the message-passing scheme to learn the representation of a node by iteratively transforming, and propagating/aggregating node features from its local neighborhoods. Along with this idea, different designs of GNN architectures have been proposed, including graph convolutional networks (GCNs) (Kipf and Welling 2017; Defferrard, Bresson, and Vandergheynst 2016), graph attention networks (GAT) (Veličković et al. 2018; Wang et al. 2019) and many others (Hamilton, Ying, and Leskovec 2017; Xu et al. 2019; Klicpera, Bojchevski, and Günnemann 2019; Wu et al. 2019; Chen et al. 2020). Despite their promising results, existing GNNs developed for *semi-supervised* node classification predominantly assume that the provided gold-labeled nodes are relatively abundant. This assumption is often impractical as data labeling requires intensive domain knowledge, especially when considering the heterogeneity of graph-structured data (Yao et al. 2020; Ding et al. 2020). When only few labeled nodes per class are available, how to improve the expressive power of Graph ML models for tackling the *few-shot semi-supervised* node classification problem remains understudied and meanwhile requires urgent research efforts.

However, it is a non-trivial and challenging task mainly because of two reasons: (i) ***overfitting and oversmoothing***. On the one hand, most of the existing GNNs own a shallow design with restricted receptive fields, thereby restricting the efficient propagation of label information (Li, Han, and Wu 2018). When training on few labeled nodes, a GNN model tends to overfit and goes timber easily. On the other hand, in order to propagate the label signals more broadly, larger receptive fields of GNNs, i.e., the number of layers, are particularly desirable (Klicpera, Bojchevski, and Günnemann 2019). Due to the entanglement of representation transformation and propagation in each layer, GNNs will face the oversmoothing issue when increasing the model depth (Liu, Gao, and Ji 2020), which in turn renders the learned node representations inseparable; (ii) ***no auxiliary knowledge.*** Though previous works proposed for graph few-shot learning (Ding et al. 2020) or cross-network transfer learning (Yao et al. 2020) also focus on related low-resource scenarios, their key enabler lies in transferring knowledge from either label-rich node classes or other similar networks. Nonetheless, such auxiliary knowledge is commonly not accessible, making those methods practically infeasible to be applied to few-shot semi-supervised learning. As suggested by previous research, pseudo-labeling (Li, Han, and Wu 2018; Sun, Lin, and Zhu 2020) is commonly beneficial to solve semi-supervised learn-

ing, whereas inaccurate pseudo labels may instead lead to abysmal failure. Hence, how to infer optimal pseudo labels on unlabeled nodes plays a pivotal role to solve the studied research problem.

To address the aforementioned challenges, we propose a new graph meta-learning framework, Meta Propagation Networks (Meta-PN), which goes beyond the canonical message-passing scheme of GNNs and learns expressive node representations in a more label-efficient way. Specifically, Meta-PN is built with two simple neural networks, i.e., *adaptive label propagator* and *feature-label transformer*, which inherently decouples the entangled propagation and transformation steps of GNNs, thereby allowing sufficient propagation of label signals without suffering the oversmoothing issue. At its core, the *adaptive label propagator* is meta-learned to adjust its propagation strategy for inferring optimal pseudo labels on unlabeled nodes, according to the feedback (i.e., the performance change on the gold-labeled nodes) from the target model *feature-label transformer*. This way the generated soft pseudo labels not only capture informative local and global structure information, but more importantly, have aligned data usage with the gold-labeled nodes. Optimizing with our proposed meta-learning algorithm, those two decoupled networks are able to reinforce each other synergistically. As a result, the target model assimilates the encoded knowledge of pseudo-labeled nodes and offers excellent performance for the semi-supervised node classification problem even if only few labeled nodes are available. In summary, the contributions of our work are as follows:

- We study the problem of semi-supervised node classification under the few-shot setting, which remains largely under-studied in the Graph ML community.
- We propose a simple yet effective graph meta-learning framework Meta-PN to solve the studied problem. The essential idea is to augment the limited labeled data via a meta-learned label propagation strategy.
- We conduct comprehensive evaluations on different graph benchmark datasets to corroborate the effectiveness of Meta-PN. The results show its superiority over the state-of-the-arts on semi-supervised node classification, especially under the low-resource setting.

2 Related Work

Graph Neural Networks. Graph neural networks (GNNs), a family of neural models for learning latent node representations in a graph, have achieved gratifying success in different graph learning tasks (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2017). Originally inspired by graph spectral theory, spectral-based graph convolutional networks (GCNs) (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2017; Wu et al. 2019) extend the convolution operation in the spectral domain to network representation learning. Among them, the model proposed by Kipf et al. (Kipf and Welling 2017) has become the most prevailing one by using a linear filter. Afterwards, spatial-based graph neural networks that follow the message-passing scheme have been extensively investigated (Hamilton, Ying, and Leskovec 2017; Veličković et al. 2018; Xu et al. 2019). Those methods follow the homophily principle (McPherson, Smith-Lovin, and Cook 2001) and learn node representations by iteratively transforming, and propagating/aggregating node features within graph neighborhoods. For example, GAT (Veličković et al. 2018) and GraphSAGE (Hamilton, Ying, and Leskovec 2017) adopt different strategies to specify fine-grained weights on neighbors when aggregating neighborhood information of a node.

Deep Graph Neural Networks. Despite the success of GNNs, the notorious over-smoothing issue can largely undermine the model performance when increasing the model depth. To counter this, researchers also try to increase the message-passing range or receptive fields of GNNs by proposing different techniques, such as adding advanced normalizations (Zhao and Akoglu 2019; Li et al. 2019a), decoupling the feature transformation and propagation steps (Wu et al. 2019; Klicpera, Bojchevski, and Günnemann 2019; Liu, Gao, and Ji 2020; Dong et al. 2021) and many others (Li, Han, and Wu 2018; Xu et al. 2018). In particular, decoupled graph neural networks have become a prevailing paradigm in the community due to their simplicity and learning efficiency (Klicpera, Bojchevski, and Günnemann 2019; Dong et al. 2021; Chien et al. 2021). For example, APPNP (Klicpera, Bojchevski, and Günnemann 2019) propagates the neural predictions via personalized PageRank, which can preserve the node's local information while increasing the receptive fields. DAGNN (Liu, Gao, and Ji 2020) decouples the propagation and transformation steps and then utilizes an adaptive adjustment mechanism to balance the information from local and global neighborhoods of each node. However, these deep GNNs are not specifically developed to tackle the low-resource settings, especially when only very few labels are available.

Graph Learning with Few Labels. For real-world graph learning tasks, the amount of gold-labeled samples is usually quite limited due to the expensive labeling cost. To improve the GNN model performance on the node classes with only few labeled nodes, graph few-shot learning (Zhou et al. 2019; Ding et al. 2020; Wang et al. 2020) and cross-network transfer learning (Yao et al. 2020; Ding et al. 2021) have been proposed to transfer the knowledge from other auxiliary data source(s). Nonetheless, for the problem of *few-shot semi-supervised* node classification, such auxiliary datasets are commonly not allowed to use. As another line of related work, Li et al. (Li, Han, and Wu 2018) combined GCNs and self-training to expand supervision signals, while M3S (Sun, Lin, and Zhu 2020) advances this idea by utilizing the clustering method to eliminate the inaccurate pseudo labels. However, those methods cannot directly address the oversmoothing issue and may suffer from inaccurate pseudo labels. By conducting meta-learning on top of a decoupled design, our approach Meta-PN achieves superior performance on few-shot semi-supervised node classification.

3 Proposed Approach

We first introduce the notations used throughout this paper. Let $G = (\mathcal{V}, \mathcal{E})$ denote an undirected graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$. $\mathcal{V}^L$ and $\mathcal{V}^U$ stand for labeled and unlabeled node set, respectively. Let n denote the number of nodes and m the number of edges. The nodes in G are described by the

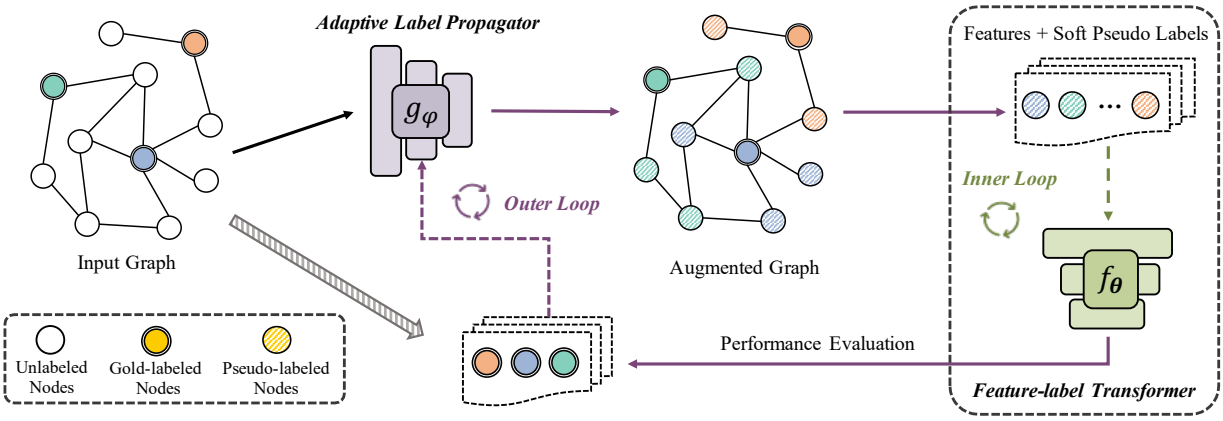


Figure 1: Illustration of our Meta-PN framework. The *adaptive label propagator* propagates known labels to unlabeled nodes and the *feature-label transformer* transforms the features of each node to a soft label vector. Specifically, the *adaptive label propagator* is meta-learned to adjust its label propagation strategy to infer accurate pseudo labels on unlabeled nodes, according to the *feature-label transformer*'s performance change on the labeled nodes. We omit the node features for simplicity.

attribute matrix $\mathbf{X} \in \mathbb{R}^{n \times f}$, where f denotes the number of features per node. The graph structure of G is described by the adjacent matrix $\mathbf{A} \in \{0,1\}^{n \times n}$, while $\tilde{\mathbf{A}}$ stands for the adjacency matrix for a graph with added self-loops. We let $\mathbf{D}$ and $\tilde{\mathbf{D}}$ be the diagonal degree matrix of $\mathbf{A}$ and $\tilde{\mathbf{A}}$, respectively. Moreover, $\tilde{\mathbf{A}}_{sym} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ denote the symmetric normalized adjacency matrix with self-loops. The class (or label) matrix is represented by $\mathbf{Y} \in \mathbb{R}^{n \times c}$, where c denotes the number of classes.

3.1 Architecture Overview

For solving the problem of few-shot semi-supervised node classification, we propose a new framework Meta Label Propagation (Meta-PN), which is built with two simple neural networks, i.e., *adaptive label propagator* and *feature-label transformer*. By decoupling the propagation and transformation steps with two independent networks, such a design inherently allows large receptive fields without suffering performance deterioration. Upon our proposed meta-learning algorithm, the meta learner – *adaptive label propagator* learns to adjust its propagation strategy for inferring accurate pseudo labels on unlabeled nodes, by using the feedback from the target model. Meanwhile, the target model – *feature-label transformer* assimilates both the structure and feature knowledge from pseudo-labeled nodes, therefore addressing the challenges behind few-shot semi-supervised learning. Specifically, we introduce the architecture details as follows:

Adaptive Label Propagator (Meta Learner). In order to enable broader propagation of label signals, we propose to adopt the idea of label propagation (LP) (Zhu and Ghahramani 2002) to encode informative local and global structural information. Similar to the message-passing scheme adopted by many GNNs, label propagation follows the principle of Homophily (McPherson, Smith-Lovin, and Cook 2001) that indicates two connected nodes tend to be similar (share same labels). Specifically, the objective of LP is to find a prediction matrix $\hat{\mathbf{Y}} \in \mathbb{R}^{n \times c}$ that agrees with the label matrix $\mathbf{Y}$ while being smooth on the graph such that nearby vertices have similar soft labels (Zhou et al. 2004). Generally, the solution can be approximated via the iteration as follows:

$$\hat{\mathbf{Y}} = \mathbf{Y}^{(K)}, \mathbf{Y}^{(k+1)} = \mathbf{T}\mathbf{Y}^{(k)}, \tag{1}$$

where $\mathbf{Y}^{(0)} = \mathbf{Y}$ and K denotes the number of power iteration (propagation) steps. The transition matrix is denoted by $\mathbf{T}$, which can be set as any form of normalized adjacency matrix (e.g., $\tilde{\mathbf{A}}_{sym}$). After K iterations of label propagation, the predicted soft label matrix $\hat{\mathbf{Y}}$ can capture the prior knowledge of neighborhood label distribution up to K hops away.

In practice, various propagation schemes can be adopted for LP, such as the Personalized PageRank (Klicpera, Bojchevski, and Günnemann 2019) where $\mathbf{Y}^{(k+1)} = (1-\alpha)\mathbf{T}\mathbf{Y}^{(k)} + \alpha\mathbf{Y}^{(0)}$. With appropriate teleport probability α, the smoothed labels can avoid losing the focus on local neighborhood even using infinitely many propagation steps (Klicpera, Bojchevski, and Günnemann 2019). However, most of the existing LP algorithms cannot adaptively balance the label information from different neighborhoods for each node, which largely restricts the model expressive power when learning with complex real-world graphs.

To counter this issue, we build an *adaptive label propagator* $g_\phi(\cdot)$ parameterized with ϕ, which is able to adjust the contribution of different propagation steps for computing the smoothed label vector of one node. Specifically, the propagation strategy can be formulated as:

$$\hat{\mathbf{Y}}_{i,:} = \sum_{k=0}^{K} \gamma_{ik}\mathbf{Y}_{i,:}^{(k)}, \mathbf{Y}^{(k+1)} = \mathbf{T}\mathbf{Y}^{(k)}, \tag{2}$$

where γ_{ik} denotes the influence from k-hop neighborhood for node v_i and can be computed by the attention mechanism:

$$\gamma_{ik} = \frac{\exp\left(\mathbf{a}^{\mathrm{T}}\mathrm{ReLU}\left(\mathbf{W}\mathbf{Y}_{i,:}^{(k)}\right)\right)}{\sum_{k'=0}^{K}\exp\left(\mathbf{a}^{\mathrm{T}}\mathrm{ReLU}\left(\mathbf{W}\mathbf{Y}_{i,:}^{(k')}\right)\right)}, \tag{3}$$

where $\mathbf{a} \in \mathbb{R}^c$ is the attention vector and $\mathbf{W} \in \mathbb{R}^{c \times c}$ is a weight matrix. By setting the attention vector and weight matrix as learnable parameters, the *adaptive label propagator* acquire the capability of adjusting its propagation strategy for each node and the final smoothed labels can capture rich structure information of the input graph.

Feature-label Transformer (Target Model). After encoding the structure knowledge into the smoothed label matrix $\hat{\mathbf{Y}}$, we then build a *feature-label transformer* $f_{\boldsymbol{\theta}}(\cdot)$ that transforms node features to node label, in order to further capture feature-based graph information. For each node v_i, the *feature-label transformer* parameterized with $\boldsymbol{\theta}$ takes the node feature vector $\mathbf{X}_{i,:}$ as input and predicts its node label $\mathbf{P}_{i,:}$ by:

$$\mathbf{P}_{i,:} = f_{\boldsymbol{\theta}}(\mathbf{X}_{i,:}), \tag{4}$$

where $f_{\boldsymbol{\theta}}(\cdot)$ is a multi-layer perceptron (MLP) followed by a softmax function.

In order to learn the target model , i.e., *feature-label transformer*, we take the soft pseudo labels computed by the *adaptive label propagator* as "ground-truth". Ideally, if the generated pseudo labels are of high quality, they can be used to augment the insufficient labeled nodes to avoid overfitting and improve the model generalization ability (Li, Han, and Wu 2018). In the meantime, high-quality pseudo-labeled data not only encodes the feature patterns of unlabeled nodes, but also carries informative local and global structure knowledge, which enables the target model to leverage larger receptive fields without suffering from performance degradation. As a result, the *feature-label transformer* can achieve excellent performance on the problem of few-shot semi-supervised node classification.

It is worth mentioning that, the target model trained with meaningful pseudo labels can be considered as a special variant of GCN, which allows far more propagation steps with much fewer parameters. Due to the space limit, we attach the detailed proof in Appendix A.1.

Learning to Propagate. One key challenge of our approach lies in how to learn a better label propagation strategy for generating pseudo labels on unlabeled nodes. If the pseudo labels are inaccurate, the target model may easily overfit to mislabeled nodes and encounter severe performance degradation (Ren et al. 2018). This issue is also known as the problem of confirmation bias in pseudo-labeling (Arazo et al. 2020). While inferring accurate pseudo labels by recursively selecting a subset of samples, re-training the prediction model will be too expensive and unstable. Hence, without linking the two networks in a principled way, it is almost infeasible to enforce the *adaptive label propagator* to efficiently infer optimal label propagation strategy for improving the performance of the *feature-label transformer*.

In this work, we propose to tackle this problem through a unified meta-learning algorithm, allowing the model to infer accurate pseudo labels for unlabeled nodes and learn a better target model. In a sense, if the generated pseudo labels are of high quality, their data utility should align with the gold-labeled nodes. Accordingly, we can derive the following meta-learning objective: *optimal pseudo labels generated by label propagation should maximize the performance (minimize the classification loss) on the small set of gold-labeled training nodes*. For each *meta label propagation* task, the goal is to generate pseudo labels for a batch of unlabeled nodes using the feedback of the target model (i.e., *feature-label transformer*). By optimizing the *adaptive label propagator* on a meta-level, the meta learner can adjust its label propagation strategy to generate informative pseudo-labeled data.

3.2 Model Learning via Bi-level Optimization

The above meta-learning objective implies a bi-level optimization problem with $\boldsymbol{\phi}$ as the outer-loop parameters and $\boldsymbol{\theta}$ as the inner-loop parameters. This problem shares the same formulation with many meta-learning algorithms that have been proposed for solving different learning tasks such as few-shot learning (Finn, Abbeel, and Levine 2017), hyper-parameter optimization (Baydin et al. 2018), and neural architecture search (Liu, Simonyan, and Yang 2018). Specifically, let $\mathcal{L}$ denote the cross-entropy loss for node classification, and this bi-level optimization problem can be formulated as:

$$\begin{aligned}&\text{Outer loop:}\\ &\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathbb{E}_{v_i \in \mathcal{V}^L}[\mathcal{L}(f_{\boldsymbol{\theta}^*(\boldsymbol{\phi})}(\mathbf{X}_{i,:}), \mathbf{Y}_{i,:})],\\ &\text{Inner loop:}\\ &\boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{v_i \in \mathcal{V}^U}[\mathcal{L}(f_{\boldsymbol{\theta}}(\mathbf{X}_{i,:}), g_{\boldsymbol{\phi}}(\mathbf{Y}, \mathbf{A})_{i,:})].\end{aligned} \tag{5}$$

The optimal solution of this bi-level optimization problem can potentially train a highly discriminative *feature-label transformer* with abundant pseudo-labeled data and only a small set of gold-labeled data. However, deriving exact solutions for this bi-level problem is indeed analytically intractable and computationally expensive, owing to the fact that it requires solving for the optimal $\boldsymbol{\theta}^*(\boldsymbol{\phi})$ whenever $\boldsymbol{\phi}$ gets updated. To approximate the optimal solution $\boldsymbol{\theta}^*(\boldsymbol{\phi})$, we propose to take one step of gradient descent update for $\boldsymbol{\theta}$, without solving the inner-loop optimization completely by training until convergence. This way allows the optimization algorithm to alternatively update the parameters of *feature-label transformer* in the inner loop and the parameters of *adaptive label propagator* in the outer loop:

Target Model (Inner-loop) Update. Given a batch of unlabeled nodes from $\mathcal{V}^U$, we update the target model parameters $\boldsymbol{\theta}$ by taking their pseudo labels computed by the *adaptive label propagator* as ground-truth. For simplicity, we use $J_{\text{pseudo}}(\boldsymbol{\theta}, \boldsymbol{\phi})$ to denote the inner-loop loss computed on a batch of pseudo-labeled nodes. Assuming that parameter $\boldsymbol{\theta}$ is updated using the computed gradient descent on $J_{\text{pseudo}}(\boldsymbol{\theta}, \boldsymbol{\phi})$, with a learning rate $\eta_{\boldsymbol{\theta}}$, then we have:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} J_{\text{pseudo}}(\boldsymbol{\theta}, \boldsymbol{\phi}). \tag{6}$$

Meta Learner (Outer-loop) Update. Note that the dependency between $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ allows us to compute the meta-level (outer-loop) loss using the gold-labeled nodes from $\mathcal{V}^L$. We denote this loss by $J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))$ for the purpose of simplicity, and back-propagate this loss to compute the gradient for the *feature-label transformer*. Having the gradient, we can update on the backward parameters $\boldsymbol{\phi}$ with learning rate $\eta_{\boldsymbol{\phi}}$:

$$\boldsymbol{\phi}' = \boldsymbol{\phi} - \eta_{\boldsymbol{\phi}} \nabla_{\boldsymbol{\phi}} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi})). \tag{7}$$

Algorithm 1: The learning algorithm of Meta-PN.

Input: The input graph $G = (\mathcal{V}, \mathcal{E})$ with labeled node set $\mathcal{V}^L$ and unlabeled node set $\mathcal{V}^U$; Batch size B
Output: The well-trained *feature-label transformer*

Initialize the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$
while *not converge* **do**
  Randomly sample a batch of B unlabeled nodes
  ▷ *Pseudo Label Generation*
  Compute the pseudo labels for sampled nodes using the *adaptive label propagator* $g_{\boldsymbol{\phi}}(\cdot)$
  ▷ *Inner-loop Update for* $\boldsymbol{\theta}$
  Compute $J_{\text{pseudo}}(\boldsymbol{\theta}, \boldsymbol{\phi})$ using the generated pseudo labeled nodes
  Update parameters $\boldsymbol{\theta}$ of the *feature-label transformer* $f_{\boldsymbol{\theta}}(\cdot)$ according to Eq. (6)
  ▷ *Outer-loop Update for* $\boldsymbol{\phi}$
  Compute $J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))$ on $\mathcal{V}^L$ using the updated *feature-label transformer*
  Update parameters $\boldsymbol{\phi}$ of the *adaptive label propagator* $g_{\boldsymbol{\phi}}(\cdot)$ according to Eq. (7) and Eq. (8)
Fine-tune the *feature-label transformer* using $\mathcal{V}^L$.
return The well-trained *feature-label transformer*

To further compute the gradient of $\boldsymbol{\phi}$, we apply chain rule to differentiate $J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))$ with respect to $\boldsymbol{\phi}$ via $\boldsymbol{\theta}'$, where $\boldsymbol{\theta}'(\boldsymbol{\phi}) = \boldsymbol{\theta} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} J_{\text{pseudo}}(\boldsymbol{\theta}, \boldsymbol{\phi})$. The full derivation is delegated to the Appendix A.2. Here, we directly present the final result:

$$\nabla_{\boldsymbol{\phi}} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi})) \approx -\frac{\eta_{\boldsymbol{\phi}}}{2\epsilon}[\nabla_{\boldsymbol{\phi}} J_{\text{pseudo}}(\boldsymbol{\theta}^+, \boldsymbol{\phi}) - \nabla_{\boldsymbol{\phi}} J_{\text{pseudo}}(\boldsymbol{\theta}^-, \boldsymbol{\phi})], \tag{8}$$

where $\boldsymbol{\theta}^{\pm} = \boldsymbol{\theta} \pm \epsilon \nabla_{\boldsymbol{\theta}'} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))$, and ϵ is a small scalar for finite difference approximation.

By alternating the update rules in Eq. (6) and Eq. (7), we are able to progressively learn the two modules. The complete meta-learning algorithm is shown in Algorithm 1. Finally, as the *feature-label transformer* only learns from unlabeled data with pseudo labels generated by the *adaptive label propagator*, we can further fine-tune the *feature-label transformer* on labeled data to improve its accuracy. After the model converges, we use the *feature-label transformer* to make final predictions on unlabeled nodes.

4 Experiments

We evaluate the effectiveness of our approach on different benchmark datasets. Specifically, our evaluation centers around three questions: (i) can Meta-PN outperform state-of-the-art GNN models when labeled data is extremely sparse? (ii) compared with the state-of-the-art GNNs, can Meta-PN achieve competitive performance under the standard semi-supervised setting? and (iii) when the data-scale goes large, how would Meta-PN perform compared to other methods?

4.1 Experimental Setup

Evaluation Datasets. We conduct experiments on five graph benchmark datasets for semi-supervised node classification to demonstrate the effectiveness of the proposed Meta-PN. The detailed statistics of the datasets are summarized in Table 1. Specifically, **Cora-ML**, **CiteSeer** (Sen et al. 2008) and **PubMed** (Namata et al. 2012) are the three most widely used citation networks. **MS-CS** is a co-authorship network based on the Microsoft Academic Graph (Shchur et al. 2018). For data splitting, we follow the previous work (Klicpera, Bojchevski, and Günnemann 2019) and split each dataset into training set (i.e., K nodes per class for K-shot task), validation set and test set. In addition, to further evaluate the performance of different methods on large-scale graphs, we further include the **ogbn-arxiv** datasets from Open Graph Benchmark (OGB) (Hu et al. 2020). For the ogbn-arxiv dataset, we randomly sample 1.0%, 1.5%, 2.0%, 2.5% nodes from its training splits as labeled data while using the same validation and test splits in OGB Benchmark (Hu et al. 2020). Note that for all the datasets, we run each experiment 100 times with multiple random splits and different initializations.

Table 1: Summary statistics of the evaluation datasets.

Dataset	# Nodes	# Edges	# Features	# Classes
Cora-ML	2,810	7,981	2,879	7
CiteSeer	2,110	3,668	3,703	6
PubMed	19,717	44,324	500	3
MS-CS	18,333	81,894	6,805	15
ogbn-arxiv	169,343	1,166,243	15	40

Compared Methods. To corroborate the effectiveness of our approach, three categories of baselines are included in our experiments: (i) *Classical Models*. **MLP**, **LP** (Label Propagation) (Zhou et al. 2004) are two classical models using only feature and structure information, respectively. **GCN** (Kipf and Welling 2017) and **SGC** (Wu et al. 2019) are two representative GNN models. Due to the space limit, we omit some baselines like GAT, GraphSAGE since similar results can be observed; (ii) *Label-efficient GNNs*. **GLP** (Generalized Label Propagation) and **IGCN** (Improved GCN) (Li et al. 2019b) are two models combine label propagation and GCN from a unifying graph filtering perspective. **M3S** (Sun, Lin, and Zhu 2020) is a multi-stage self-training framework, which incorporates self-supervised learning to improve the model performance with few labeled nodes; (iii) *Deep GNNs*. **APPNP** (Klicpera, Bojchevski, and Günnemann 2019) decouples prediction and propagation with performing personalized propagation of neural predictions, while **DAGNN** (Liu, Gao, and Ji 2020) adaptively incorporate information from large receptive fields. **C&S** (Huang et al. 2021) is an effective model that combines label propagation and simple neural networks. **GPR-GNN** (Chien et al. 2021) addresses the limitation of APPNP on different types of graphs with adaptive propagation weights.

Implementation Details. All our experiments are conducted with a 12 GB Ti-tan Xp GPU. The proposed Meta-PN is implemented in PyTorch. We use a 2-layer MLP with 64 hidden units for the feature-label transformer. We apply L2 regularization with $\lambda = 0.005$ on the weights of the first neural layer and set the dropout rate for both neural layers to be

Table 2: Test accuracy on few-shot semi-supervised node classification: mean accuracy (%) ± 95% confidence interval.

Method	Cora-ML		CiteSeer		PubMed		MS-CS	
	3-shot	5-shot	3-shot	5-shot	3-shot	5-shot	3-shot	5-shot
MLP	41.07 ± 0.76	51.12 ± 0.61	43.34 ± 0.56	44.90 ± 0.60	56.59 ± 0.93	59.90 ± 0.84	70.33 ± 0.37	79.41 ± 0.31
LP	62.07 ± 0.71	68.01 ± 0.62	54.07 ± 0.59	55.73 ± 1.19	58.75 ± 0.89	59.91 ± 0.85	57.96 ± 0.69	62.98 ± 0.61
GCN	48.02 ± 0.89	67.32 ± 1.02	53.60 ± 0.86	62.60 ± 0.58	58.89 ± 0.80	65.77 ± 0.98	69.24 ± 0.94	84.43 ± 0.89
SGC	49.60 ± 0.55	67.24 ± 0.86	57.37 ± 0.98	61.55 ± 0.53	63.37 ± 0.93	64.93 ± 0.81	72.11 ± 0.76	87.51 ± 0.27
GLP	65.57 ± 0.26	71.26 ± 0.31	65.76 ± 0.49	71.36 ± 0.18	65.34 ± 0.54	65.26 ± 0.29	86.10 ± 0.21	86.94 ± 0.23
IGCN	66.60 ± 0.29	72.50 ± 0.20	67.47 ± 0.29	72.92 ± 0.10	62.28 ± 0.23	65.19 ± 0.13	85.83 ± 0.06	87.01 ± 0.05
M3S	64.66 ± 0.31	69.64 ± 0.18	65.12 ± 0.20	68.18 ± 0.18	63.40 ± 0.32	68.85 ± 0.26	84.96 ± 0.18	86.83 ± 0.29
APPNP	72.39 ± 0.98	78.32 ± 0.58	67.55 ± 0.77	71.08 ± 0.61	70.52 ± 0.62	74.24 ± 0.87	86.65 ± 0.42	90.13 ± 0.86
DAGNN	71.86 ± 0.75	77.20 ± 0.69	66.62 ± 0.27	70.55 ± 0.12	71.22 ± 0.82	73.91 ± 0.71	86.32 ± 0.57	90.30 ± 0.66
C&S	68.93 ± 0.68	73.37 ± 0.24	63.02 ± 0.72	64.72 ± 0.53	70.51 ± 0.57	73.22 ± 0.57	85.86 ± 0.45	87.99 ± 0.24
GPR-GNN	70.98 ± 0.84	75.18 ± 0.52	64.32 ± 0.81	65.28 ± 0.52	71.03 ± 0.73	74.08 ± 0.65	86.12 ± 0.37	90.29 ± 0.38
Meta-PN	**74.94 ± 0.25**	**79.88 ± 0.15**	**70.48 ± 0.34**	**74.14 ± 0.50**	**73.25 ± 0.77**	**77.78 ± 0.92**	**88.99 ± 0.29**	**91.31 ± 0.22**
	79.85	82.37	70.98	73.77	69.50	78.13	84.21	91.62

0.3. For methods based on label propagation, we use $K = 10$ power iteration (propagation) steps by default. To make a fair comparison, we let all the configurations of the baselines be the same as Meta-PN including neural network layers, hidden units, regularization, propagation steps, early stopping and initialization. We use Adam to optimize the baseline methods as suggested and fine-tune for the corresponding learning rate on different datasets. More details on model implementation and parameter selection can be found in Appendix A.3.

4.2 Evaluation Results

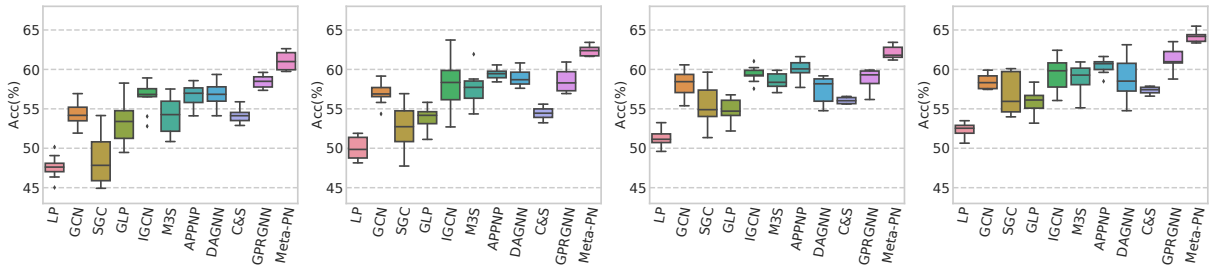
Few-shot Semi-supervised Evaluation. First, we evaluate the proposed approach Meta-PN and all the baseline methods on few-shot semi-supervised node classification, which aims to predict the missing node labels with only a few labeled nodes. The average test accuracies under the few-shot setting (i.e., 3-shot and 5-shot) can be found in Table 2. Additional results are provided in Appendix A.4 due to the space limit. From the reported results, we can clearly see that Meta-PN significantly outperforms all the baseline methods on each dataset based on paired t-tests with $p < 0.05$. Specifically, we elaborate our in-depth observations and analysis as follows: (i) without abundant labeled data, classical models including vanilla GNNs only obtain very poor classification accuracy under different evaluation entries; (ii) overall the label-efficient GNNs outperform classical GNNs, but still cannot achieve satisfying results. One major reason is that those methods cannot handle the oversmoothing issue since they are incapable of explicitly leveraging the knowledge from large receptive fields; (iii) by enabling better propagation of label signals, deep GNNs have stronger performance than both the classical models and label-efficient GNNs, which again demonstrates the necessity of addressing the oversmoothing issue for solving the few-shot semi-supervised learning problem. However, existing deep GNNs are not specifically developed to tackle the data sparsity issue, thus their performance still falls behind Meta-PN by a noticeable margin on different datasets when only very few labels are available. This observation proves that Meta-PN is able to address the overfitting and oversmoothing issues when labeled data is extremely sparse by combining the power of large receptive fields and pseudo labels.

Table 3: Test accuracy on standard semi-supervised node classification: mean accuracy (%) ± 95% confidence interval.

Method	Cora-ML	CiteSeer	PubMed	MS-CS
MLP	$68.42 \pm .34$	$63.98 \pm .44$	$69.47 \pm .47$	$88.30 \pm .13$
LP	$75.74 \pm .27$	$65.62 \pm .43$	$69.82 \pm .70$	$72.03 \pm .25$
GCN	$82.70 \pm .37$	$73.62 \pm .39$	$76.84 \pm .44$	$91.10 \pm .20$
SGC	$75.97 \pm .72$	$75.57 \pm .28$	$71.24 \pm .86$	$90.56 \pm .14$
GLP	$81.67 \pm .14$	$75.21 \pm .14$	$78.95 \pm .09$	$91.85 \pm .04$
IGCN	$82.11 \pm .09$	$75.22 \pm .10$	$79.06 \pm .07$	$91.60 \pm .03$
M3S	$82.72 \pm .13$	$73.73 \pm .32$	$77.62 \pm .11$	$91.08 \pm .09$
APPNP	$85.09 \pm .25$	$75.73 \pm .30$	$79.73 \pm .31$	$91.74 \pm .16$
DAGNN	$85.65 \pm .23$	$74.53 \pm .17$	$79.59 \pm .37$	$92.80 \pm .17$
C&S	$83.18 \pm .31$	$70.51 \pm .24$	$77.10 \pm .34$	$92.49 \pm .19$
GPR-GNN	$83.53 \pm .31$	$71.18 \pm .25$	$79.62 \pm .46$	$92.57 \pm .21$
Meta-PN	**$86.33 \pm .36$**	**$77.13 \pm .31$**	**$80.39 \pm .53$**	**$93.92 \pm .17$**

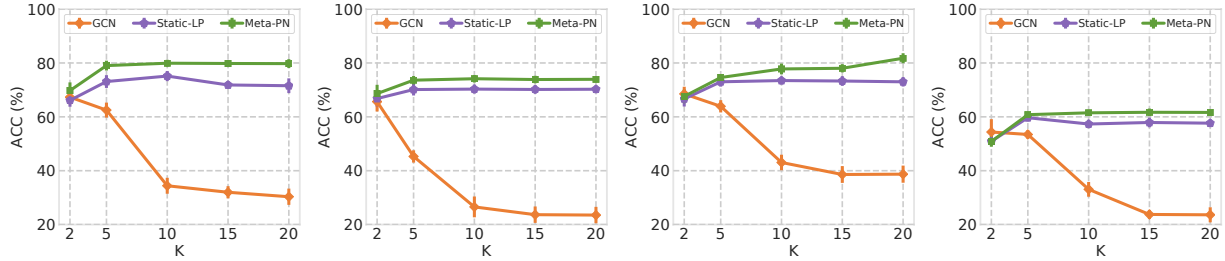
Standard Semi-supervised Evaluation. To make our evaluation more comprehensive, we then examine the effectiveness of Meta-PN under the standard semi-supervised node classification tasks. As in (Klicpera, Bojchevski, and Günnemann 2019), we randomly sample 20 labeled nodes for each class (i.e., 20-shot) as the training set. According to the average performance reported in Table 3, we make the following observations: (i) the GNN models which combine both the structure and feature knowledge from labeled nodes can obtain improved node classification performance compared to methods which only consider feature or structure information individually; (ii) under the standard semi-supervised node classification task, the performance of the label-efficient GNNs are close to vanilla GNNs; (iii) though Meta-PN is mainly proposed for few-shot semi-supervised learning, it still achieves the best performance for the standard semi-supervised node classification task, illustrating the superiority of our graph approach.

Evaluation on Open Graph Benchmark (OGB). Real-world graphs commonly have a larger size and more node classes than many toy graphs, leading to the collected graphs having noisy structures and complex properties. To further



(a) label ratio = 1.0% (b) label ratio = 1.5% (c) label ratio = 2.0% (d) label ratio = 2.5%

Figure 2: Comparison results on ogbn-arxiv w.r.t different size of training labels.



(a) Cora-ML (b) CiteSeer (c) PubMed (d) Ogbn-arxiv

Figure 3: Few-shot (i.e., 5-shot or 1.0% label ratio) evaluation on different datasets w.r.t. propagation steps (K).

illustrate the effectiveness of our approach on large-scale real-world graphs, we adopt the widely used ogbn-arxiv dataset and compare all the methods under the few-shot setting (i.e., from 1% to 2.5% label ratio). We summarize their performance for few-shot semi-supervised node classification on ogbn-arxiv in Figure 2 by changing the ratio of training labels, in which we omit MLP as its test accuracy is much lower than the other methods. We can observe that Meta-PN can significantly outperform all the baseline models under different few-shot environments. Compared to the other baseline methods, the performance of Meta-PN is relatively stable when we decrease the ratio of training labels, which demonstrates the robustness of Meta-PN in handling noisy and complex real-world graphs. Remarkably, our approach can achieve close performance to the vanilla GCN on ogbn-arxiv with much fewer labeled nodes (2.5% vs. 54%).

Parameter & Ablation Analysis. To demonstrate the effects of using different propagation steps and the importance of the meta-leaned label propagation strategy for Meta-PN, we compare our approach with two baselines under the 5-shot (or 1.0% label ratio for ogbn-arxiv) semi-supervised setting with varying number of propagation steps. Specifically, *GCN* learns the node representation with the standard message-passing scheme while *Static-LP* representing the variant of Meta-PN that uses fixed teleport probabilities instead of meta-learned ones. The evaluation results are shown in Figure 3. As we can observe from the figure, *GCN* can achieve very close performance with the other two methods when the number of propagation steps is relatively small. While if we largely increase the number of propagation steps, the performance of *GCN* breaks down due to the oversmoothing issue. Empowered by the idea of label propagation, *Static-LP* can largely alleviate the oversmoothing issue and significantly outperform *GCN*. This verifies that larger propagation steps or receptive fields are necessary for improving the performance of GNN when labeled data is extremely limited. In the meantime, *Static-LP* still falls behind Meta-PN, mainly because of the infeasibility of balancing the importance of different receptive fields. On the contrary, Meta-PN is able to address this issue by inferring optimal pseudo labels on unlabeled nodes with our meta-learning algorithm. Its performance becomes stable when $K \geq 10$, indicating that Meta-PN can obtain good performance considering both efficiency and effectiveness with a moderate number of propagation steps (e.g., $K = 10$).

5 Conclusion

In this paper, we propose a new graph meta-learning framework, Meta Propagation Networks (Meta-PN), for solving the problem of few-shot semi-supervised node classification. Based on the meta-learned label propagation strategy, we are able to generate informative pseudo labels on unlabeled nodes, in order to augment the insufficient labeled data and learn a powerful GNN model. Though built with simple neural networks, Meta-PN effectively enables larger receptive fields and avoids oversmoothing when learning with very few labeled data. We test Meta-PN on a spectrum of benchmark datasets and the results well demonstrate its effectiveness. For future work, it would be interesting to investigate other pseudo-labeling strategies for solving the studied problem.

Acknowledgements

This work is partially supported by Office of Naval Research (ONR) N00014-21-1-4002 and Army Research Office (ARO) W911NF2110030.

References

Arazo, E.; Ortego, D.; Albert, P.; O'Connor, N. E.; and McGuinness, K. 2020. Pseudo-labeling and confirmation bias in deep semi-supervised learning. In *IJCNN*.

Baydin, A. G.; Cornish, R.; Rubio, D. M.; Schmidt, M.; and Wood, F. 2018. Online Learning Rate Adaptation with Hypergradient Descent. In *ICLR*.

Chen, M.; Wei, Z.; Huang, Z.; Ding, B.; and Li, Y. 2020. Simple and deep graph convolutional networks. In *ICML*.

Chien, E.; Peng, J.; Li, P.; and Milenkovic, O. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *ICLR*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*.

Ding, K.; Wang, J.; Li, J.; Shu, K.; Liu, C.; and Liu, H. 2020. Graph prototypical networks for few-shot learning on attributed networks. In *CIKM*.

Ding, K.; Zhou, Q.; Tong, H.; and Liu, H. 2021. Few-shot Network Anomaly Detection via Cross-network Meta-learning. In *The Web Conference*.

Dong, H.; Chen, J.; Feng, F.; He, X.; Bi, S.; Ding, Z.; and Cui, P. 2021. On the Equivalence of Decoupled Graph Convolution Network and Label Propagation. In *The Web Conference*.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*.

Huang, Q.; He, H.; Singh, A.; Lim, S.-N.; and Benson, A. R. 2021. Combining Label Propagation and Simple Models Out-performs Graph Neural Networks. In *ICLR*.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *NeurIPS*.

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*.

Klicpera, J.; Groß, J.; and Günnemann, S. 2019. Directional Message Passing for Molecular Graphs. In *ICLR*.

Li, G.; Muller, M.; Thabet, A.; and Ghanem, B. 2019a. Deepgcns: Can gcns go as deep as cnns? In *CVPR*.

Li, Q.; Han, Z.; and Wu, X.-M. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.

Li, Q.; Wu, X.-M.; Liu, H.; Zhang, X.; and Guan, Z. 2019b. Label efficient semi-supervised learning via graph filtering. In *CVPR*.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. DARTS: Differentiable Architecture Search. In *ICLR*.

Liu, M.; Gao, H.; and Ji, S. 2020. Towards deeper graph neural networks. In *KDD*.

McPherson, M.; Smith-Lovin, L.; and Cook, J. M. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology*.

Namata, G.; London, B.; Getoor, L.; Huang, B.; and EDU, U. 2012. Query-driven active surveying for collective classification. In *Workshop on MLG*.

Ren, M.; Zeng, W.; Yang, B.; and Urtasun, R. 2018. Learning to reweight examples for robust deep learning. In *ICML*.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine*.

Shchur, O.; Mumme, M.; Bojchevski, A.; and Günnemann, S. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*.

Sun, K.; Lin, Z.; and Zhu, Z. 2020. Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes. In *AAAI*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *ICLR*.

Wang, N.; Luo, M.; Ding, K.; Zhang, L.; Li, J.; and Zheng, Q. 2020. Graph Few-shot Learning with Attribute Matching. In *CIKM*.

Wang, Q.; Mao, Z.; Wang, B.; and Guo, L. 2017. Knowledge graph embedding: A survey of approaches and applications. In *TKDE*.

Wang, X.; Ji, H.; Shi, C.; Wang, B.; Ye, Y.; Cui, P.; and Yu, P. S. 2019. Heterogeneous graph attention network. In *The Web Conference*.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *ICML*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How powerful are graph neural networks? In *ICLR*.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*.

Yao, H.; Zhang, C.; Wei, Y.; Jiang, M.; Wang, S.; Huang, J.; Chawla, N.; and Li, Z. 2020. Graph few-shot learning via knowledge transfer. In *AAAI*.

Zafarani, R.; Abbasi, M. A.; and Liu, H. 2014. *Social media mining: an introduction*. Cambridge University Press.

Zhao, L.; and Akoglu, L. 2019. PairNorm: Tackling Oversmoothing in GNNs. In *ICLR*.

Zhou, D.; Bousquet, O.; Lal, T. N.; Weston, J.; and Schölkopf, B. 2004. Learning with local and global consistency. In *NeurIPS*.

Zhou, F.; Cao, C.; Zhang, K.; Trajcevski, G.; Zhong, T.; and Geng, J. 2019. Meta-GNN: On Few-shot Node Classification in Graph Meta-learning. In *CIKM*.

Zhu, X.; and Ghahramani, Z. 2002. Learning from labeled and unlabeled data with label propagation. *Technical Report*.

A Appendix

A.1 Theoretical Analysis of Meta-PN

As we introduced in Section 3.1, the target model trained with meaningful pseudo labels can be considered as a special variant of GCN, which allows far more propagation steps with much fewer parameters. Here we present the theoretical analysis to show this connection. For simplicity, we first consider the 1-layer (one propagation step) case, and we have the cross-entropy loss function as follows:

$$\begin{aligned}\mathcal{L}_1 &= \mathcal{L}_{\text{CE}}(\mathbf{T}f_{\boldsymbol{\theta}_1}(\mathbf{X}), \mathbf{Y}) \\ &= -\sum_{j\in\mathcal{V}^L, k\in\mathcal{C}} y_{j,k}(\log \sum_{i\in\mathcal{V}} t_{j,i}p_{i,k}),\end{aligned} \tag{9}$$

where $\mathbf{p}_i$ represents the result from the transformation function $f_{\boldsymbol{\theta}_1}(\mathbf{x}_i)$ of GCN and $p_{i,k}$ denotes its k-th element. $\mathcal{L}_{\text{CE}}$ denotes the cross-entropy loss. The gradients of the objective function with respect to $\boldsymbol{\theta}_1$ can be written as:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_1}\mathcal{L}_1 &= -\sum_{j\in\mathcal{V}^L, k\in\mathcal{C}} y_{j,k}\nabla_{\boldsymbol{\theta}_1}(\log \sum_{i\in\mathcal{V}} t_{j,i}p_{i,k}) \\ &= -\sum_{j\in\mathcal{V}^L, k\in\mathcal{C}} y_{j,k}\frac{\sum_{i\in\mathcal{V}} t_{j,i}\nabla_{\boldsymbol{\theta}_1}p_{i,k}}{\sum_{q\in\mathcal{V}} t_{j,q}p_{q,k}}.\end{aligned} \tag{10}$$

Note that $\mathbf{y}_j$ is an one-hot vector, only the $h(j)$-th element is non-zero. Then the gradients can be rewritten as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_1}\mathcal{L}_1 &= -\sum_{j\in\mathcal{V}^L} y_{j,h(j)}\frac{\sum_{i\in\mathcal{V}} t_{j,i}\nabla_{\boldsymbol{\theta}_1}p_{i,h(j)}}{\sum_{q\in\mathcal{V}} t_{j,q}p_{q,h(j)}} \\ &= -\sum_{i\in\mathcal{V}, j\in\mathcal{V}^L} \frac{t_{j,i}p_{i,h(j)}}{\sum_{q\in\mathcal{V}} t_{j,q}p_{q,h(j)}} y_{j,h(j)}\frac{\nabla_{\boldsymbol{\theta}_1}p_{i,h(j)}}{p_{i,h(j)}} \\ &= \sum_{i\in\mathcal{V}, j\in\mathcal{V}^L} \frac{t_{j,i}p_{i,h(j)}}{\sum_{q\in\mathcal{V}} t_{j,q}p_{q,h(j)}} \nabla_{\boldsymbol{\theta}_1}\mathcal{L}_{\text{CE}}(\mathbf{p}_i, \mathbf{y}_j).\end{aligned} \tag{11}$$

If we train a neural network $f_{\boldsymbol{\theta}_2}(\cdot)$ on the directly propagated pseudo labels $\hat{\mathbf{Y}} = \mathbf{TY}$, the gradients of its objective function can be computed as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_2}\mathcal{L}_2 &= \nabla_{\boldsymbol{\theta}_2}\mathcal{L}_{\text{CE}}(f_{\boldsymbol{\theta}_2}(\mathbf{X}), \mathbf{TY}) \\ &= \sum_{i\in\mathcal{V}, k\in\mathcal{C}} \sum_{j\in\mathcal{V}^L} t_{i,j}y_{j,k}\nabla_{\boldsymbol{\theta}_2}\log p_{i,k} \\ &= \sum_{i\in\mathcal{V}, j\in\mathcal{V}^L} t_{i,j}\nabla_{\boldsymbol{\theta}_2}\sum_{k\in\mathcal{C}} y_{j,k}\log p_{i,k} \\ &= \sum_{i\in\mathcal{V}, j\in\mathcal{V}^L} t_{i,j}\nabla_{\boldsymbol{\theta}_2}\mathcal{L}_{\text{CE}}(\mathbf{p}_i, \mathbf{y}_j).\end{aligned} \tag{12}$$

As we can see, the learning process of $f_{\boldsymbol{\theta}_2}(\cdot)$ is equivalent to $f_{\boldsymbol{\theta}_1}(\cdot)$ by ignoring the regularization term $\frac{p_{i,h(j)}}{\sum_{q\in\mathcal{V}} t_{j,q}p_{q,h(j)}}$ in Eq. (11). For Meta-PN, this regularization term is replaced by the teleport probability α, thus the learned target model is essentially a special case of GCN.

A.2 Computing $\nabla_{\boldsymbol{\phi}} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))$

Due to the relationship between $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ as shown in Eq. (5), $\nabla_{\boldsymbol{\phi}} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))$ is differentiable with respect to $\boldsymbol{\phi}$. Then we first compute the gradient $\nabla_{\boldsymbol{\phi}} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))$ by applying chain rule with implicit and explicit gradients as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\phi}} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi})) &= \nabla_{\boldsymbol{\phi}} J_{\text{gold}}(\boldsymbol{\theta} - \eta_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}} J_{\text{pseudo}}(\boldsymbol{\theta}, \boldsymbol{\phi})) \\ &= -\eta_{\boldsymbol{\theta}}\nabla^2_{\boldsymbol{\theta},\boldsymbol{\phi}} J_{\text{pseudo}}(\boldsymbol{\theta}, \boldsymbol{\phi})\nabla_{\boldsymbol{\theta}'} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi})),\end{aligned} \tag{13}$$

where $\nabla_{\boldsymbol{\phi}} J^+_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))$ is the implicit gradient that assumes all other variables except $\boldsymbol{\phi}$ as constants.

The second term of the above equation contains an expensive matrix-vector product, here we use the finite difference approximation to reduce the complexity. Let ϵ denote a small constant number, according to the finite difference method, $\frac{\partial f(x,z)}{\partial x} * k \approx \lim_{\epsilon\to 0}\frac{f(x+\epsilon,z)-f(x-\epsilon,z)}{2\epsilon} * k = \lim_{\epsilon\to 0}\frac{f(x+k\epsilon,z)-f(x-k\epsilon,z)}{2k\epsilon} * k = \lim_{\epsilon\to 0}\frac{f(x+k\epsilon,z)-f(x-k\epsilon,z)}{2\epsilon}$. Thus we can get:

$$\begin{aligned}&\nabla^2_{\boldsymbol{\theta},\boldsymbol{\phi}} J_{\text{pseudo}}(\boldsymbol{\theta}, \boldsymbol{\phi})\nabla_{\boldsymbol{\theta}'} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi})) \\ &= \nabla_{\boldsymbol{\phi}}\Big(\nabla_{\boldsymbol{\theta}} J_{\text{pseudo}}(\boldsymbol{\theta}, \boldsymbol{\phi}) * \nabla_{\boldsymbol{\theta}'} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))\Big) \\ &\approx \lim_{\epsilon\to 0}\frac{\nabla_{\boldsymbol{\phi}} J_{\text{pseudo}}\Big(\boldsymbol{\theta} + \nabla_{\boldsymbol{\theta}'} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))\epsilon, \boldsymbol{\phi}\Big)}{2\epsilon} \\ &\quad - \frac{\nabla_{\boldsymbol{\phi}} J_{\text{pseudo}}\Big(\boldsymbol{\theta} - \nabla_{\boldsymbol{\theta}'} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))\epsilon, \boldsymbol{\phi}\Big)}{2\epsilon}.\end{aligned} \tag{14}$$

With Eq. (13) and Eq. (14), we can get:

$$\begin{aligned}\nabla_{\boldsymbol{\phi}} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi})) &= -\eta_{\boldsymbol{\phi}}\lim_{\epsilon\to 0}\frac{\nabla_{\boldsymbol{\phi}} J_{\text{pseudo}}(\boldsymbol{\theta}^+, \boldsymbol{\phi}) - \nabla_{\boldsymbol{\phi}} J_{\text{pseudo}}(\boldsymbol{\theta}^-, \boldsymbol{\phi})}{2\epsilon} \\ &= -\frac{\eta_{\boldsymbol{\phi}}}{2\epsilon}[\nabla_{\boldsymbol{\phi}} J_{\text{pseudo}}(\boldsymbol{\theta}^+, \boldsymbol{\phi}) - \nabla_{\boldsymbol{\phi}} J_{\text{pseudo}}(\boldsymbol{\theta}^-, \boldsymbol{\phi})],\end{aligned} \tag{15}$$

where $\boldsymbol{\theta}^{\pm} = \boldsymbol{\theta} \pm \nabla_{\boldsymbol{\theta}'} J_{\text{gold}}(\boldsymbol{\theta}'(\boldsymbol{\phi}))\epsilon$.

A.3 Implementation Details

Meta-PN. We implement the proposed Meta-PN in PyTorch. We set the batch size to 1,024 for Cora-ML and Citeseer, and 4,096 for the other datasets. Specifically, we use two-layer MLP with 64 hidden units for the *feature-label transformer* and optimize it with Adam. We grid search for the learning rate $\eta_{\boldsymbol{\theta}}$ in $\{1\times10^{-5}, 5\times10^{-5}, 1\times10^{-4}, 5\times10^{-4}, 1\times10^{-3}, 5\times10^{-3}, 1\times10^{-2}, 5\times10^{-2}, 1\times10^{-1}, 5\times10^{-1}\}$. Meanwhile, we optimize the *adaptive label propagator* with Adam and grid search for the learning rate $\eta_{\boldsymbol{\phi}}$ in $\{1\times10^{-5}, 5\times10^{-5}, 1\times10^{-4}, 5\times10^{-4}, 1\times10^{-3}, 5\times10^{-3}, 1\times10^{-2}, 5\times10^{-2}, 1\times10^{-1}, 5\times10^{-1}\}$. We also search for dropout rate in $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7\}$. The optimal values are selected when the model achieve the best performance for validation set. Following (Klicpera, Bojchevski, and Günnemann 2019), the early stopping criterion uses a patience of $p = 100$ and an (unreachably high) maximum of $n = 10,000$ epochs. The patience is reset whenever the accuracy increases or the loss decreases on the validation set.

Baselines. In our experiments, we compare our approach with different methods including MLP, LP, GCN, SGC, GLP,

IGCN, M3S, APPNP, DAGNN, C&S and GPR-GNN. For the baseline methods, we adopt their public implementations and the details are as follows:

- **MLP**: For a fair comparison, we use a 2-layer fully connected network with 64 hidden units for representation learning.
- **LP** (Zhou et al. 2004): We use the same propagation step K and the teleport probability as Meta-PN for a fair comparison.
- **GCN**[1] (Kipf and Welling 2017): We build the GCN model with two graph convolutional layers (64 dimensions) for learning node representations.
- **SGC**[2] (Wu et al. 2019): After the feature pre-processing step, it learns the node representations with 2-layer feature propagation with 64 hidden units.
- **GLP & IGCN**[3] (Li et al. 2019b): It uses a two-layer structure (64 hidden units) in which the filter parameters k and α is set to be 5 and 10 for 20-shot, and is set to be 10 and 20 for all the other tasks. The results with the best performing filter (i.e., RNM or AR) are reported.
- **M3S**[4] (Sun, Lin, and Zhu 2020): We fix the number of clusters as 200 and select the best number of layers and stages as suggested by the authors.
- **APPNP**[5] (Klicpera, Bojchevski, and Günnemann 2019): Similar to Meta-PN, we use the 2-layer MLP (64 hidden units), with 10 steps of propagation. For the best performance, we set the teleport probability $\alpha = 0.1$ for the citation graphs and use $\alpha = 0.2$ for the co-authorship graph due to their structural difference.
- **DAGNN**[6] (Liu, Gao, and Ji 2020): We let the size of hidden unit and the propagation step to be the same as Meta-PN for fairness.
- **C&S**[7] (Huang et al. 2021): We use the MLP base predictor and follow the default settings provided by the authors for the best performance.
- **GPR-GNN**[8] (Chien et al. 2021): For fair comparison, we use the random walk path lengths with K = 10 and use a 2-layer (MLP) with 64 hidden units for the neural network component.

For all the baseline methods, we use Adam as optimizer and fine-tune the hyperparameters on each dataset. Specifically, we grid search for the learning rate in $\{1 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}, 5 \times 10^{-2}, 1 \times 10^{-1}, 5 \times 10^{-1}\}$ and dropout rate in $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7\}$. Also, we use the same early stop strategy as for Meta-PN.

[1]https://github.com/tkipf/pygcn
[2]https://github.com/Tiiiger/SGC
[3]https://github.com/liqimai/Efficient-SSL
[4]https://github.com/datake/M3S
[5]https://github.com/klicperajo/ppnp
[6]https://github.com/mengliu1998/DeeperGNN
[7]https://github.com/CUAI/CorrectAndSmooth
[8]https://github.com/jianhao2016/GPRGNN

Packages Used for Implementation. For reproducibility, we also list the packages we use in the implementation with their corresponding versions: python==3.6.6, pytorch==1.4.0, cuda==10.1, numpy==1.19.2, ogb==1.3.1 and scikit-learn==0.24.0.

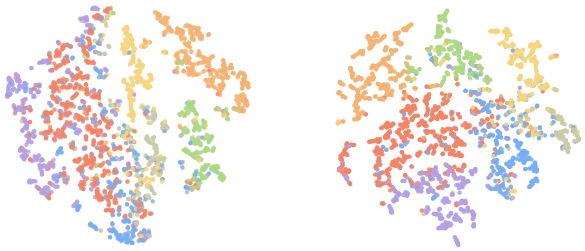
A.4 Additional Experimental Results

As a supplement to Table 2, we compare Meta-PN with the baseline methods on one more low-resource semi-supervised node classification task (10-shot) and the test results are summarized in Table 4. Based on the results, we can observe that the proposed Meta-PN can significantly outperform all the baseline methods for the 10-shot task on different datasets, which further illustrates the effectiveness of Meta-PN for low-resource semi-supervised node classification.

Table 4: Test accuracy on 10-shot semi-supervised node classification with different models: Mean accuracy (%) $\pm$ 95% confidence interval.

	Cora-ML	CiteSeer	PubMed	MS-CS
MLP	$60.68 \pm .48$	$56.15 \pm .69$	$63.97 \pm .73$	$85.67 \pm .24$
LP	$72.38 \pm .34$	$59.97 \pm .62$	$65.69 \pm .98$	$66.27 \pm .22$
GCN	$78.48 \pm .43$	$69.04 \pm .74$	$70.88 \pm .75$	$89.39 \pm .19$
SGC	$74.40 \pm .66$	$72.79 \pm .27$	$68.34 \pm .96$	$89.01 \pm .31$
GLP	$76.83 \pm .25$	$71.97 \pm .16$	$73.65 \pm .19$	$90.11 \pm .17$
IGCN	$78.32 \pm .20$	$\underline{73.32 \pm .11}$	$74.45 \pm .24$	$88.26 \pm .12$
M3S	$77.34 \pm .25$	$70.08 \pm .19$	$72.78 \pm .27$	$89.01 \pm .23$
APPNP	$\underline{82.21 \pm .24}$	$72.70 \pm .49$	$75.01 \pm .67$	$\underline{91.66 \pm .11}$
DAGNN	$80.80 \pm .31$	$72.72 \pm .35$	$\underline{76.70 \pm .61}$	$91.60 \pm .07$
C&S	$75.38 \pm .31$	$69.61 \pm .39$	$74.47 \pm .18$	$90.71 \pm .29$
GPR-GNN	$78.82 \pm .33$	$69.83 \pm .32$	$74.75 \pm .26$	$91.48 \pm .14$
Meta-PN	$\mathbf{83.84 \pm .29}$	$\mathbf{75.30 \pm .42}$	$\mathbf{78.44 \pm .41}$	$\mathbf{92.26 \pm .17}$

Embedding Visuliazation. To show the quality of the embedding from Meta-PN, we use t-SNE to visualize the extracted node representations from a strong baseline APPNP and Meta-PN for comparison. With the node's color denoting its label, from Figure 4 we can observe that though APPNP can effectively identify some of the classes, the boundary between different classes is still unclear. The proposed approach Meta-PN is able to generate more compact and separated clusters, which again verifies its superiority.



(a) APPNP (b) Meta-PN

Figure 4: Embedding visualization results of APPNP and Meta-PN on the Cora-ML dataset.