Learning How to Propagate Messages in Graph Neural Networks

Teng Xiao[§*], Zhengyu Chen[†§], Donglin Wang[§‡], and Suhang Wang[*]
[§]Machine Intelligence Lab (MiLAB), AI Division, School of Engineering, Westlake University
[†]College of Computer Science & Technology, Zhejiang University [*]The Pennsylvania State University
tvx5054@psu.edu,chenzhengyu@zju.edu.cn,wangdonglin@westlake.edu.cn,szw494@psu.edu

ABSTRACT

This paper studies the problem of learning message propagation strategies for graph neural networks (GNNs). One of the challenges for graph neural networks is that of defining the propagation strategy. For instance, the choices of propagation steps are often specialized to a single graph and are not personalized to different nodes. To compensate for this, in this paper, we present learning to propagate, a general learning framework that not only learns the GNN parameters for prediction but more importantly, can explicitly learn the interpretable and personalized propagate strategies for different nodes and various types of graphs. We introduce the optimal propagation steps as latent variables to help find the maximum-likelihood estimation of the GNN parameters in a variational Expectation-Maximization (VEM) framework. Extensive experiments on various types of graph benchmarks demonstrate that our proposed framework can significantly achieve better performance compared with the state-of-the-art methods, and can effectively learn personalized and interpretable propagate strategies of messages in GNNs.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

Graph Neural Networks; Graph Representation Learning

ACM Reference Format:
Teng Xiao, Zhengyu Chen, Donglin Wang, and Suhang Wang. 2021. Learning How to Propagate Messages in Graph Neural Networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467451

1 INTRODUCTION

Graphs are ubiquitous in the real world, such as social networks, knowledge graphs, and molecular structures. Recently, Graph Neural Networks (GNNs) have achieved state-of-the-art performance across various tasks on graphs, such as semi-supervised node classification [25, 39, 48] and link prediction [18]. Typically, GNNs exploit message propagation strategies to learn expressive node representations by propagating and aggregating the messages between neighboring nodes. Various message propagation layers have been proposed, including graph convolutional layers (GCN) [25], graph attention layers (GAT) [39], and many others [7, 11, 18, 26, 40, 45]. Recent studies [5, 25, 28] show that GNNs suffer from the over-smoothing issue (the representations of nodes are inclined to converge to a certain value, making the model performance degrade significantly by stacking too many propagation layers).

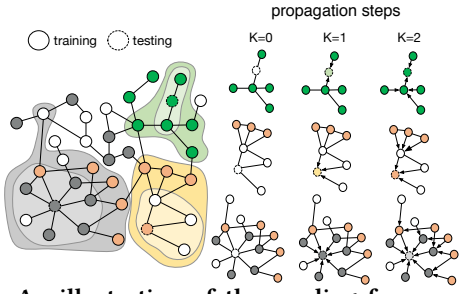

Figure 1: An illustration of the needing for personalized propagation. Color (green, gray, and yellow) denotes the class of the node. White nodes denote the unlabeled nodes.

To tackle the over-smoothing issue, many efforts have been taken [6, 25, 27, 45, 47]. For example, several works [25, 27, 45] try to add residual or dense connections in the message propagation layer to preserve the locality. Although the convergence speed of over-smoothing is retarded, most of these methods do not really outperform 2-layer models such as GCN or GAT. A crucial question remains to be addressed in order to make GNN a success: *Do we really need a very deep GNN? Furthermore, can we automatically choose propagation steps to specific nodes and graphs?*

In this paper, we provide answers to both questions. Our key insight is that different nodes and various types of graphs may need different propagation steps to accurately predict node labels. As suggested by [7, 38, 45], low-degree nodes only have a small number of neighbors, which receive very limited information from neighborhoods and deep GNNs may perform well on those nodes. In contrast, nodes with higher degrees are more likely to suffer from over-smoothing. Figure 1 illustrates the key motivation of this paper. Intuitively, the propagation step smoothes the features locally along the edges of the graph and ultimately encourages similar predictions among locally connected nodes. We can obviously observe that the optimal propagation step for the test green and yellow class nodes is two, however, the optimal step for the black class node is one since more steps will bring the noise to the representation of it. Thus, it is a natural idea that different propagation patterns tend to work better for different nodes. For different types of graphs, the number of propagation steps is also different. For example, for

[‡]Corresponding author.

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467451

those heterophily graphs [34] (where connected nodes may have different class labels and dissimilar features), message propagation steps may hurt the node similarity, and stacking deep layers cannot achieve better performance compared with homophily graphs [7, 22]. Since there is no strategy to learn how to propagate the message, existing GNNs need a hand-crafted layer number depending on different types of nodes and graphs. This requires expert domain knowledge and careful parameter tuning and will be sub-optimal. However, whether it is possible to learn personalized strategies while optimizing GNNs remains an open problem.

Motivated by the discussion above, in this paper, we investigate whether one can automatically learn personalized propagation strategies to help GNNs learn interpretable prediction and improve generalization. In essence, we are faced with several challenges: (i) The graph data is very complex, thus building hand-crafted and heuristic rules for propagation steps for each node tends to be infeasible when we know little knowledge underlying graph or the node distributions are too complicated. (ii) In practice, there is no way to directly access the optimal strategy of propagation. The lack of supervision about how to propagate obstructs models from modeling the distribution of propagation for each node. (iii) GNNs are also prone to be over-fitting [37], where the GNNs fit the training data very well but generalizes poorly to the testing data. It will suffer from the over-fitting issue more severely when we utilize an addition parameterized model for each node to learn how to propagate given limited labeled data in the real world.

To address the challenges mentioned above, we propose a simple yet effective framework called learning to propagate (L2P) to simultaneously learn the optimal propagation strategy and GNN parameters to achieve personalized and adaptive propagation. Our framework requires no heuristics and is generalizable to various types of nodes and graphs. Since there is no supervision, we adopt the principle of the probabilistic generative model and introduce the optimal propagation steps as latent variables to help find the maximum-likelihood estimation of GNN parameters in a variational Expectation-Maximization (VEM) framework. To further alleviate the over-fitting, we introduce an efficient bi-level optimization algorithm. The bi-level optimization closely matches the definition of generalization since validation data can provide accurate estimation of the generalization. The main contributions of this work are:

- We study a new problem of learning propagation strategies for GNNs. To address this problem, we propose a general L2P framework which can learn personalized and interpretable propagation strategies, and achieve better performance simultaneously.
- We propose an effective stochastic algorithm based on variational inference and bi-level optimization for the L2P framework, which enables simultaneously learning the optimal propagation strategies and GNN parameters, and avoiding the over-fitting issue.
- We conduct experiments on homophily and heterophily graphs and the results demonstrate the effectiveness of our framework.

2 RELATED WORK

2.1 Graph Neural Networks

GNNs have achieved great success in modeling graph-structured data. Generally, GNNs can be categorized into two categories, i.e., spectral-based and spatial-based. Spectral-based GNNs define graph convolution based on spectral graph theory [4, 12, 40]. GCN [25] further simplifies graph convolutions by stacking layers of first-order Chebyshev polynomial filters together with some approximations. Spatial-based methods directly define updating rules in the spatial space. For instance, GAT [39] introduces the self-attention strategy into aggregation to assign different importance scores of neighborhoods. We refer interested readers to the recent survey [41] for more variants of GNN architectures. Despite the success of variants GNNs, the majority of existing GNNs aggregate neighbors' information for representation learning, which are shown to suffer from the over-smoothing [28, 35] issue when many propagation layers are stacked, the representations of all nodes become the same.

To tackle the over-smoothing issue, some works [25, 27] try to add residual or dense connections [45] in propagation steps for preserving the locality of the node representations. Other works [6, 37] augment the graph by randomly removing a certain number of edges or nodes to prevent the over-smoothing issue. Recently, GCNII [7] introduces initial residual and identity mapping techniques for GCN and achieves promising performance. Since the feature propagation and transformation steps are commonly coupled with each other in standard GNNs, several works [26, 30] separate this into two steps to reduce the risk of over-smoothing. We differ from these methods as (1) instead of focus on alleviating over-smoothing, we argue that different nodes and graphs may need a different number of propagation layers, and propose a framework of learning propagation strategies that generalizable to various types of graphs and backbones, and (2) we propose the bilevel optimization to utilize validation error to guide learning propagation strategy for improving the generalization ability of graph neural networks.

2.2 The Bi-level Optimization

Bi-level optimization [31], which performs upper-level learning subject to the optimality of lower-level learning, has been applied to different tasks such as few-shot learning [8, 9, 14], searching architectures [29], and reinforcement learning [49]. For the graph domain, Franceschi et al. propose a bi-level optimization objective to learn the structures of graphs. Some works [17, 51] optimize a bi-level objective via reinforcement learning to search the architectures of GNNs. Moreover, Meta-attack [53] adopts the principle of meta-learning to conduct the poisoning attack on the graphs by optimizing a bi-level objective. Recently, Hwang et al. propose SELAR [20] which learns the weighting function for self-supervised tasks to help the primary task on the graph with a bi-level objective. To conduct the few-shot learning in graphs, the work [30], inspired by MAML [14], try to obtain a parameter initialization that can adapt to unseen tasks quickly, using gradients information from the bi-level optimization. By contrast, in this paper, our main concern is the generalization, and we propose a bilevel programming with variational inference to develop a framework for learning propagation strategies, while avoiding the over-fitting issues.

3 PRELIMINARIES

3.1 Notations and Problem Definition

Let $G = (\mathcal{V}, \mathcal{E})$ denote a graph, where $\mathcal{V}$ is a set of $|\mathcal{V}| = N$ nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of $|\mathcal{E}|$ edges between nodes. $\mathbf{A} \in \{0, 1\}^{N \times N}$ is the adjacency matrix of G. The (i, j)-th element $\mathbf{A}_{ij} = 1$ if there

exists an edge between node v_i and v_j, otherwise $\mathbf{A}_{ij} = 0$. Furthermore, we use $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{N \times d}$ to denote the features of nodes, where $\mathbf{x}_n$ is the d-dimensional feature vector of node v_n. Following the common semi-supervised node classification setting [25, 39], only a small portion of nodes $\mathcal{V}_o = \{v_1, v_2, \ldots, v_o\}$ are associated with observed labels $\mathcal{Y}^o = \{y_1, y_2, \ldots, y_o\}$, where y_n denotes the label of v_n. $\mathcal{V}_u = \mathcal{V} \backslash \mathcal{V}_o$ is the set of unlabeled nodes. Given the adjacency matrix $\mathbf{A}$, features $\mathbf{X}$ and the observed labels $\mathcal{Y}^o$, the task of node classification is to learn a function f_θ which can accurately predict the labels $\mathcal{Y}^u$ of unlabeled nodes $\mathcal{V}_u$.

3.2 Message Propagation

Generally, GNNs adopt the message propagation process, which iteratively aggregates the neighborhood information. Formally, the propagation process of the k-th layer in GNN is two steps:

$$\mathbf{m}_{k,n} = \text{AGGREGATE}\left(\left\{\mathbf{h}_{k-1,u} : u \in \mathcal{N}(n)\right\}\right) \tag{1}$$

$$\mathbf{h}_{k,n} = \text{UPDATE}\left(\mathbf{h}_{k-1,n}, \mathbf{m}_{k,n}, \mathbf{h}_{0,n}\right) \tag{2}$$

where $\mathcal{N}_n$ is the set of neighbors of node v_n, AGGREGATE is a permutation invariant function. After K message-passing layers, the final node embeddings $\mathbf{H}_K$ are used to perform a given task. In general, most state-of-the-art GNN backbones [7, 25, 26, 39] follow this message propagation form with different AGGREGATE functions, UPDATE function, or initial feature $\mathbf{h}_{0,n}$. For instance, APPNP [26] and GCNII [7] add the initial feature $\mathbf{h}_{0,n} = \text{MLP}(\mathbf{x}_n; \theta)$ to each layer in the UPDATE function. In general, GNN consists of several message propagation layers. We abstract the message propagation with K layers as one parameterized function $GNN(\mathbf{X}, \mathbf{A}, K)$.

4 LEARNING TO PROPAGATE

In this section, we introduce the Learning to Propagate (L2P) framework, which can perform personalized message propagation for better node representation learning and a more interpretable prediction process. The key idea is to introduce a discrete latent variable t_n for each node v_n, which denotes the personalized optimal propagation step of v_n. How to learn t_n is challenging given no explicit supervision on the optimal propagation step of v_n. To address the challenge, we propose a generative model for modeling the joint distribution of node labels and propagation steps conditioned on node attributes and graphs, i.e., $p(y_n, t_n|\mathbf{X}, \mathbf{A})$, and formulate the Learning to Propagate framework as a variational objective, where the goal is to find the parameters of GNNs and the optimal propagation distribution, by iteratively approximating and maximizing the log-likelihood function. To alleviate the over-fitting issue, we further frame the variational process as a bi-level optimization problem, and optimize the variational parameters of learning the propagation strategies in an outer loop to maximize generalization performance of GNNs trained based on the learned strategies.

4.1 The Generative Process

Generally, we can consider the designing process of graph neural networks as follows: we first choose the number of propagation layers K for all nodes and the type of the aggregation function parameterized by θ. Then for each training label y_n of node n, we typically conduct the Maximum Likelihood Estimation (MLE) of the marginal log-likelihood over the observed labels as:

$$\max_\theta \mathcal{L}(\theta; \mathbf{A}, \mathbf{X}, \mathcal{Y}^o) = \sum\nolimits_{y_n \in \mathcal{Y}^o} \log p_\theta(y_n|GNN(\mathbf{X}, \mathbf{A}, K)), \tag{3}$$

where $p_\theta(y_n|GNN(\mathbf{X}, \mathbf{A}, K)) = p(y_n|\mathbf{H}_K)$ is the predicted probability of node v_n having label y_n using $\mathbf{H}_{K,n}$. $\mathbf{H}_{K,n}$ is the node representation of v_n after stacking K propagation steps (see § 3.2). Generally, a softmax is applied on $\mathbf{H}_{K,n}$ for predicting label y_n.

Although the message propagation strategy above has achieved promising performance, it has two drawbacks: (i) The above strategy treats each node equally, i.e., each node stacks K-layers; while in practice, different nodes may need different propagation steps/layers. Simply using a one-for-all strategy could potentially lead to sub-optimal decision boundaries and is less interpretable, and (ii) Different datasets/graphs may also have different optimal propagation steps. Existing GNNs require a hand-crafted number of propagation steps, which requires expert domain knowledge, careful parameter tuning, and is time-consuming. Thus, it would be desirable if we could learn the personalized and adaptive propagation strategy which is applicable to various types of graphs and GNN backbones.

Based on the motivation above, we propose to learn a personalized propagate distribution from the given labeled nodes and utilize the learned propagate distribution at test time, such that each test node would automatically find the optimal propagate step to explicitly improve the performance. A natural idea of learning optimal propagate distribution is supervised learning. However, there is no direct supervision of the optimal propagate strategy for each node. To solve this challenge, we treat the optimal propagation layer of each node as a discrete latent variable and adopt the principle of the probabilistic generative model, which has shown to be effective in estimating the underlying data distribution [10, 33, 42, 44].

Specifically, for each node v_n, we introduce a latent discrete variable $t_n \in \{0, 1, 2, \cdots, K\}$ to denote its optimal propagation step, where K is the predefined maximum step. Note that t_n can be 0, which corresponds to use non-aggregated features for prediction. We allow t_n to be 0, because for some nodes in heterophily graphs, the neighborhood information is noisy, aggregating the neighborhood information may result in worse performance [22, 36]. t_n is node-wise because the optimal propagation step for each node may vary largely from one node to another. With the latent variable $\{t_n\}_{n=1}^{|\mathcal{V}|}$, we propose the following generative model with modeling the joint distribution of each observed label y_n and latent t_n:

$$p_\theta(y_n, t_n|\mathbf{X}, \mathbf{A}) = p_\theta(y_n|GNN(\mathbf{X}, \mathbf{A}, t_n))p(t_n), \tag{4}$$

where $p(t_n)$ is the prior of propagation variable and θ is the parameter shared by all nodes. $p_\theta(\mathbf{y}_n|GNN(\mathbf{X}, \mathbf{A}, t_n))$ represents the label prediction probability using v_n's representation from the t_n-th layer, i.e., $\mathbf{H}_{t_n,n}$. Since we do not have any prior of how to propagate, $p(t_n) = \frac{1}{K+1}$ is defined as uniform distribution on all layers of all nodes in this paper. We can also use an alternative prior with lower probability on the deeper layers, if we want to encourage shallower GNNs. Given the generative model in Eq. (4) and from the Bayesian perspective, what we are interested in are two folds: (1) Learning the parameter θ of the GNN by maximizing the follow likelihood which helps make label prediction in the testing phase:

$$\log p_\theta(y_n|\mathbf{X}, \mathbf{A}) = \log \sum\nolimits_{t_n=0}^{K} p_\theta(y_n|GNN(\mathbf{X}, \mathbf{A}, t_n))p(t_n). \tag{5}$$

(2) Inferring the following posterior $p(t_n|\mathbf{X}, \mathbf{A}, y_n)$ of the latent variable t, which is related to the optimal propagation distribution.

$$p(t_n = k|\mathbf{X}, \mathbf{A}, y_n) = \frac{p_\theta(y_n|GNN(\mathbf{X}, \mathbf{A}, k))}{\sum_{k'=0}^{K} p_\theta(y_n|GNN(\mathbf{X}, \mathbf{A}, k'))}. \quad (6)$$

Intuitively, this posterior can be understood as we choose the propagation step t_n of node v_n according to the largest likelihood (i.e., the smallest loss) in the defined propagation steps.

However, there are several challenges to solve these two problems. For learning, we cannot directly learn the parameter θ, since it involves marginalizing the latent variable, which is generally time-consuming and intractable [24]. In terms of the inference, since we do not have labels for test nodes, i.e, y_n for $v_n \in \mathcal{V}_u$, the non-parametric true posterior in Eq. (6), which involves evaluating the likelihood $p_\theta(y_n|GNN(\mathbf{X}, \mathbf{A}, k))$ of test nodes, is not applicable. To solve the challenges in the learning and inference, we adopt the variational inference principle [24, 43], and instead consider the following lower bound of the marginal log-likelihood in Eq. (5) which gives rise to our following formal variational objective:

$$\mathcal{L}(\theta, q) = \mathbb{E}_{q(t_n)}\left[\log p_\theta\left(\mathrm{y}_n|\mathbf{X}, \mathbf{A}, t_n\right)\right] - \mathrm{KL}(q(t_n)||p(t_n)), \quad (7)$$

where the derivations are given in Appendix A.1 and $q(t_n)$ is the introduced variational distribution. Maximizing the ELBO $\mathcal{L}(\theta, q)$ is equivalent to (i) maximize Eq. (5) and to (ii) make the variational distributions $q(t_n)$ of each node be close to its intractable true posteriors $p(t_n|\mathbf{X}, \mathbf{A}, y_n)$. Note that the ELBO holds for any type of variational distribution $q(t_n)$. We defer discussion of the learning and inference process until the next section. Here, we first introduce two ways to show how to exactly parameterize the variational distribution $q(t_n)$, resulting in two instances of our L2P framework.

4.2 Learning to Select

In the variational inference principle, we can introduce a variational distribution $q_\phi(t_n|\boldsymbol{\nu}_n)$ parameterized by $\boldsymbol{\nu}_n \in \mathbb{R}^K$. However, we cannot fit each $q_\phi(t_n|\boldsymbol{\nu}_n)$ individually by solving $N \cdot K$ parameters, which increases the over-fitting risk given the limited labels in the graphs. Thus, we consider the amortization inference [24] which avoids the optimization of the parameter $\boldsymbol{\nu}_n$ for each local variational distribution $q_\phi(t_n|\boldsymbol{\nu}_n)$. Instead, it fits a shared neural network to calculate each local parameter $\boldsymbol{\nu}_n$. Since the latent variable t_n is a discrete multinomial variable, the simplest and most naive way to represent categorical variable is the softmax function. Thus, we pass the features of nodes through a softmax function to parameterize the categorical propagation distribution as:

$$q_\phi(t_n = k|\mathbf{X}, \mathbf{A}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{H}_{k,n})}{\sum_{k'=0}^{K} \exp(\mathbf{w}_{k'}^\top \mathbf{H}_{k',n})}, \quad (8)$$

where $\mathbf{w}_k$ represents the trainable linear transformation for the k-th layer. $\mathbf{H}_{k,n}$ is the representation of node n at the k-th layer and ϕ represents the set of parameters. The main insight behind this amortization is to reuse the propagation representation of each layer, leveraging the accumulated knowledge of representation to quickly infer propagation distribution. With amortization, we reduce the number of parameters to $(K + 1) \cdot D$, where K is the predefined maximum propagation step and D is the dimension of the representation of nodes. Since this formulation directly models the selection probability overall propagation steps of each node, we refer to this method as *Learning to Select* (L2S). Figure 2(b) gives an illustration of L2S. We adopt the node representation of v_n in each layer to calculate $q_\phi(t_n = k|\mathbf{X}, \mathbf{A})$, which makes it able to personally and adaptively decide which propagation layer is best for v_n. It also allows each graph to learn its own form of propagation with its own decay form from the validation signal (see § 5.1 for details).

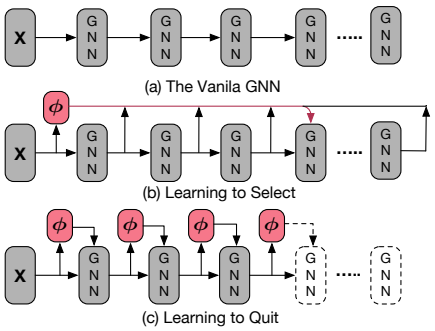

Figure 2: Illustrations of our L2P framework. (a) The vanilla GNN architecture. (b) L2S predicts the selection probability over all propagation steps for each node. (c) L2Q forces each node to personally quit its propagation process.

4.3 Learning to Quit

Instead of directly modeling the selection probability over every propagation step, we can model the probability of exiting the propagation process and transform the modeling of multinomial probability parameters into the modeling of the logits of binomial probability parameters. More specifically, we consider modeling the quit probability at each propagation layer for each node n as follows:

$$\alpha_{k,n} = \frac{1}{1 + \exp(-\mathbf{w}_k^\top \mathbf{H}_{k,n})}, \quad (9)$$

where $\alpha_{k,n}$ denotes the probability that node v_n quits propagating at the k-th layer. The challenge is how to transfer the logits $\alpha_{k,n}$ to the multinomial variational distribution $q_\phi(t_n|\mathbf{X}, \mathbf{A})$. In this paper, we consider the stick breaking (a non-parametric Bayesian) process [23] to address this challenge. Specifically, the probability of the first step (no propagation), i.e., $q(t_n = 0)$ is modeled as a break of proportion $\alpha_{0,n}$ (quit at the first-step), while the length of the remainder of the propagation is left for the next break. Each probability of propagation step can be deterministically computed by the quit probability $q(t_n = k) = \alpha_{k,n} \prod_{k'=0}^{k-1} (1 - \alpha_{k',n})$ until $K - 1$, and the probability of last propagation step is $q(t_n = K) = \prod_{k'=0}^{K-1}(1 - \alpha_{k',n})$. Assume the maximum propagation step $K = 2$, then the propagation probability is generated by 2 breaks where $p(t_n = 0) = \alpha_{0,n}$, $p(t_n = 1) = \alpha_{1,n}\left(1 - \alpha_{0,n}\right)$ and the last propagation step $p(t_n = 2) = (1 - \alpha_{1,n})\left(1 - \alpha_{0,n}\right)$ (not quit until the end). Hence, for different values of K, this non-parametric breaking process always satisfies $\sum_{k=0}^{K} q(t_n = k) = 1$. We call this method *Learning to Quit* (L2Q). Compared with L2S, L2Q models the quit probability of each node at each propagation step via the stick-breaking process which naturally induces the sparsity property of the modeling propagation step for each node. The deeper layers are less likely to be sampled. Figure 2(c) shows the architecture of L2Q.

4.4 Learning and Inference

Maximizing the ELBO in Eq. (7) is challenging. The lack of labels for test data further exacerbates the difficulty. Thus, in this paper, we propose two algorithms: the alternate expectation maximization

and iterative variational inference algorithms to maximize it.

Alternate expectation maximization. Minimization of the negative ELBO in Eq. (7) can be solved by the expectation maximization (EM) algorithm, which iteratively infers $q(t_n)$ at E-step and learns θ at M-step. More specifically, at each iteration i, given the current status parameters $\theta^{(i)}$, the E-step that maximizes $\mathcal{L}(\theta^{(i)}, q)$ w.r.t q has a closed-form solution the same as Eq. (6):

$$q^{(i+1)}(t_n) = q(t_n|\mathbf{X}, \mathbf{A}, y_n) = \frac{p_{\theta^{(i)}}(y_n|\mathbf{X}, \mathbf{A}, t_n)}{\sum_{t_n=0}^{K} p_{\theta^{(i)}}(y_n|\mathbf{X}, \mathbf{A}, t_n)}. \quad (10)$$

However, we can not utilize this non-parametric posterior since the label y_n is not available for the test nodes. We need to let the training and testing pipeline be consistent. Thus, we consider projecting the non-parametric posterior to a parametric posterior $q_\phi(t_n|\mathbf{X}, \mathbf{A})$ (i.e., L2S or L2Q). We adopt an approximation, which has also been used in the classical wake-sleep algorithm [19] by minimizing the forward KL divergence $KL(q^{(i+1)}(t_n)||q_\phi(t_n|\mathbf{X}, \mathbf{A}))$. Then we can get the following pseudo maximizing likelihood objective:

$$\phi^{(i+1)} = \arg\max_{\phi} \mathbb{E}_{q^{(i+1)}(t_n)} \left[\log q_\phi(t_n|\mathbf{X}, \mathbf{A})\right]. \quad (11)$$

Given the parametric posterior $q_{\phi^{(i+1)}}(t_n|\mathbf{X}, \mathbf{A})$, the M-step optimizes $\mathcal{L}(\theta, q_{\phi^{(i+1)}}(t_n|\mathbf{X}, \mathbf{A}))$ w.r.t θ. Since there is no analytical solution for deep neural networks, we update the model parameters θ with respect to the ELBO by one step of gradient descent.

Iterative variational inference. Although the alternate expectation maximization algorithm is effective to infer the optimal propagation variable, the alternate EM steps are time-consuming and we need calculating the loss at every layer for each training node, i.e., the $O(N * (K + 1))$ complexity. Thus, we propose an end-to-end iterative algorithm to minimize negative ELBO. Specifically, we introduce the parameterized posterior $q_\phi(t_n|\mathbf{X}, \mathbf{A})$ (i.e., L2S or L2Q) into Eq. (7) and directly optimize ELBO using reparameterization trick [24]. We infer the optimal propagation distribution $q_\phi(t_n|\mathbf{X}, \mathbf{A})$ and learn GNN weights θ jointly through standard back-propagation from the ELBO in Eq. (7). However, the optimal propagation steps t is discrete and non-differentiable which makes direct optimization difficult. Therefore, we adopt Gumbel-Softmax Sampling [21, 32], which is a simple yet effective way to substitutes the original non-differentiable sample from a discrete distribution with a differentiable sample from a corresponding Gumbel-Softmax distribution. Specifically, we minimize the following negative ELBO in Eq. (7) with the reparameterization trick [24]:

$$\mathcal{L}(\theta, \phi) = -\log p_\theta(\mathbf{y}|GNN(\mathbf{X}, \mathbf{A}, \hat{t})) + \mathrm{KL}(q_\phi(t_n|\mathbf{X}, \mathbf{A})||p(t_n)), \quad (12)$$

where $\hat{t}$ is drawn from a categorical distribution with the discrete variational distribution $q_\phi(t_n|\mathbf{X}, \mathbf{A})$ parameterized by ϕ:

$$\hat{t}_k = \frac{\exp((\log(q_\phi(t_n|\mathbf{X}, \mathbf{A})[a_k]) + g_k)/\gamma_g)}{\sum_{k'=0}^{K} \exp((\log(q_\phi(t_n|\mathbf{X}, \mathbf{A})[a_{k'}]) + g_{k'})/\gamma_g)}, \quad (13)$$

where $\{g_{k'}\}_{k'=0}^{K}$ are i.i.d. samples drawn from the Gumbel $(0, 1)$ distribution, γ_g is the softmax temperature, $\hat{t}_k$ is the k-th value of sample $\hat{t}$ and $q_\phi(t_n|\mathbf{X}, \mathbf{A})[a_k]$ indicates the a_k-th index of $q_\phi(t_n|\mathbf{X}, \mathbf{A})$, i.e., the logit corresponding the $(a_k - 1)$-th layer. Clearly, when $\tau > 0$, the Gumbel-Softmax distribution is smooth so ϕ can be optimized by standard back-propagation. The KL term in Eq. (12) is respect to two categorical distributions, thus it has a closed form.

5 BI-LEVEL VARIATIONAL INFERENCE

So far, we have proposed the L2P framework and shown how to solve it via variational inference. However, as suggested by previous work [13, 37], GNNs suffer from over-fitting due to the scarce label information in the graph domain. In this section, we propose the bilevel variational inference to alleviate the over-fitting issue.

5.1 The Bi-level Objective

For our L2P framework, the introduced inference network for joint learning optimal propagation steps in L2S and L2Q also increases the risk of over-fitting as shown in experiments (§ 6.4). To solve the over-fitting issue, we draw inspiration from gradient-based meta-learning (learning to learn) [15, 31]. Briefly, the objective of ϕ is to maximize the ultimate measure of the performance of GNN model $p_\theta(y|GNN(\mathbf{X}, \mathbf{A}, t))$, which is the model performance on a held-out validation set. Formally, this goal can be formulated as the following bi-level optimization problem:

$$\min_{\phi} \mathcal{L}_{\text{val}}\left(\theta^*(\phi), \phi\right) \text{ s.t. } \theta^*(\phi) = \arg\min_{\theta} \mathcal{L}_{\text{train}}(\theta, \phi), \quad (14)$$

where $\mathcal{L}_{\text{val}}\left(\theta^*(\phi), \phi\right)$ and $\mathcal{L}_{\text{train}}(\theta, \phi)$ are called upper-level and lower-level objectives on validation and training sets, respectively. For our L2P framework, the objective is the negative ELBO $\mathcal{L}(\theta, \phi)$ in Eq. (7). This bi-level update is to optimize the propagation strategies of each node so that the GNN model performs best on the validation set. Instead of using fixed propagation steps, it learns to assign adaptive steps while regularizing the training of a GNN model to improve the generalization. Generally, the bi-level optimization problem has to solve each level to reach a local minimum. However, calculating the optimal ϕ requires two nested loops of optimization, i.e., we need to compute the optimal parameter $\theta^*(\phi)$ for each ϕ. Thus, in order to control the computational complexity, we propose an approximate alternating optimization method by updating θ and ϕ iteratively in the next section.

5.2 Bi-level Training Algorithm

Indeed, in general, there is no closed-form expression of θ, so it is not possible to directly optimize the upper-level objective function in Eq. (14). To tackle this challenge, we propose an alternating approximation algorithm to speed up computation in this section.

Updating lower level θ. Instead of solving the lower level problem completely per outer iteration, we fix ϕ and only take the following gradient steps over mode parameter θ at the i-th iteration:

$$\theta^{(i)} = \theta^{(i-1)} - \eta_\theta \nabla_\theta \mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)}), \quad (15)$$

where η_θ is the learning rate for θ.

Updating upper level ϕ. After receiving the parameter $\theta^{(i)}$ (a reasonable approximation of $\theta^{(*)}(\phi)$), we can calculate the upper level objective, and update ϕ through:

$$\phi^{(i)} = \phi^{(i-1)} - \eta_\phi \nabla_\phi \mathcal{L}_{\text{val}}(\theta^{(i)}, \phi^{(i-1)}). \quad (16)$$

Note that $\theta^{(i)}$ is a function of ϕ due to Eq. (15), we can directly back-propagate the gradient through $\theta^{(i)}$ to ϕ. the $\nabla_\phi \mathcal{L}_{\text{val}}(\theta^{(i)}, \phi^{(i-1)})$ can be approximated as (see Appendix A.2 for detailed derivations):

$$\nabla_\phi \mathcal{L}_{\text{val}}(\theta^{(i)}, \phi^{(i-1)}) = \nabla_\phi \mathcal{L}_{\text{val}}(\bar{\theta}^{(i)}, \phi^{(i-1)}) \quad (17)$$
$$- \eta_\theta \frac{1}{\epsilon}(\nabla_\phi \mathcal{L}_{\text{train}}(\theta^{(i-1)} + \epsilon v, \phi^{(i-1)}) - \nabla_\phi \mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)})),$$

where $v = \nabla_\theta \mathcal{L}_{\text{val}}(\theta^{(i)}, \bar{\phi}^{(i-1)})$, and $\bar{\theta}^{(i)}$ and $\bar{\phi}^{(i-1)}$ means stopping the gradient. This can be easily implemented by maintaining a shadow version of $\theta^{(i-1)}$ at last step, catching the training loss $\mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)})$ and computing the new loss $\mathcal{L}_{\text{train}}(\theta^{(i-1)} + \epsilon v, \phi^{(i-1)})$. When η_θ is set to 0 in Eq. (17), the second-order derivative will disappear, resulting in a first-order approximate. In experiments in § 6.4, we study the effect of bi-level optimization, and the first- and second-order approximates.

Given the above derivations of gradients, we have the complete L2P algorithm by alternating the update rules in Eqs. (15) and (16). The time complexity mainly depends on the bi-level optimization. For the first-order approximate, the complexity is the same as vanilla GNN methods. L2P needs approximately $3\times$ training time for the second-order approximate since it needs extra forward and backward passes of the weight to compute the bilevel gradient. However, as the experiments in § 6.4 show, the first-order approximate is sufficient to achieve the best performance.

6 EXPERIMENT

In this section, we conduct experiments to evaluate the effectiveness of the proposed frameworks with comparison to state-of-the-art GNNs. Specifically, we aim to answer the following questions:

- **(RQ 1)** How effective is the proposed L2P framework for the node classification task on both heterophily and homophily graphs?
- **(RQ 2)** Could the proposed L2P alleviate over-smoothing?
- **(RQ 3)** How do the proposed learning algorithms work? Could the bi-level optimization alleviate the over-fitting issue?
- **(RQ 4)** Could the proposed framework adaptively learn propagation strategies for better understanding the graph structure?
- **(RQ 5)** Could the proposed L2P framework effectively the personalized and interpretable propagation strategies for each node?

6.1 Experimental Settings

Datasets. We conduct experiments on both homophily and heterophily graphs. For homophily graphs, we adopts three standard citation networks for semi-supervised node classification, i.e., Cora, CiteSeer, and PubMed [46]. Recent studies [7, 22, 36] show that the performance of GNNs can significantly drop on heterophily graphs, we also include heterophily benchmark in our experiments, including Actor, Cornell, Texas, and Wisconsin [7, 36]. The descriptions and statistics of these datasets are provided in Appendix A.3.

Baselines. To evaluate the effectiveness of the proposed framework, we consider the following representative and state-of-the-art GNN models on the semi-supervised node classification task. GCN [25], GAT [39], JK-Net [45], APPNP [26], DAGNN [50], IncepGCN [37], and GCNII* [7]. We also compare our proposed methods with GCN(DropEdge), ResGCN(DropEdge), JKNet(DropEdge) and IncepGCN(DropEdge) by utilizing the drop-edge trick [37]. The details and implementations of baselines are given in Appendix A.4.

Setup. For our L2P framework, we consider APPNP as our backbone unless otherwise stated, but note that our framework is broadly applicable to more complex GNN backbones [7, 25, 39]. We randomly initialize the model parameters. We utilize the first-order approximate for our methods due to its efficiency and study the effect of second-order approximate separately in § 6.4. For each search of hyper-parameter configuration, we run the experiments with 20 random seeds and select the best configuration of hyper-parameters based on average accuracy on the validation set. Hyper-parameter settings and the splitting of datasets are given in Appendix A.5.

Table 1: Summary of results on homophily graphs. Note our results can be easily improved by using a more complex backbone. For example, by using GCNII* as our backbone, L2S can achieve 85.6 ± 0.2 on Cora and 80.9 ± 0.3 on PubMed.

Method	Cora	CiteSeer	PubMed
GCN	81.3 ± 0.8	71.1 ± 0.7	78.8 ± 0.6
GAT	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
APPNP	83.3 ± 0.5	71.8 ± 0.5	79.7 ± 0.3
JKNet	80.6 ± 0.5	69.6 ± 0.2	77.8 ± 0.3
JKNet(Drop)	83.0 ± 0.3	72.2 ± 0.7	78.9 ± 0.4
Incep(Drop)	83.0 ± 0.5	72.3 ± 0.4	79.3 ± 0.3
DAGNN	84.2 ± 0.5	73.3 ± 0.6	80.3 ± 0.4
GCNII*	**85.3 ± 0.2**	73.2 ± 0.8	80.3 ± 0.4
L2S	84.9 ± 0.3	74.2 ± 0.5	80.2 ± 0.5
L2Q	85.2 ± 0.5	**74.6 ± 0.4**	**80.4 ± 0.4**

Table 2: Node classification accuracy on heterophily graphs.

Method	Actor	Cornell	Texas	Wisconsin
GCN	26.86	52.71	52.16	45.88
GAT	28.45	54.32	58.38	49.41
Geom-GCN-I	29.09	56.76	57.58	58.24
Geom-GCN-P	31.63	60.81	67.57	64.12
Geom-GCN-S	30.30	55.68	59.73	56.67
APPNP	32.41	73.51	65.41	69.02
JKNet	27.41	57.30	56.49	48.82
JKNet(Drop)	29.21	61.08	57.30	50.59
Incep(Drop)	30.13	61.62	57.84	50.20
GCNII*	35.18	76.49	77.84	81.57
L2S	36.58	80.54	84.12	84.31
L2Q	**36.97**	**81.08**	**84.56**	**84.70**

6.2 RQ1. Performance Comparison

To answer RQ1, we conduct experiments on both homophily and heterophily graphs with comparison to state-of-the-art methods.

Performance on homophily graphs. Table 1 reports the mean classification accuracy with the standard deviation on the test nodes after 10 runs. From Table 1, we have the following findings: (1) Our L2S and L2Q improve the performance of the APPAP backbone consistently and significantly in all settings. This is because that our framework has the advantage of adaptively learning personalized strategies via bi-level training. This observation demonstrates our motivation and the effectiveness of our framework. (2) Our L2S and L2Q can achieve comparable performance with state-of-the-art methods such as DAGNN and GCNII* on Cora and PubMed, and outperform them on CiteSeer. This once again demonstrates the effectiveness of our L2P framework on the node classification task. (3) In terms of our methods, the L2Q performs better than L2S, indicting that the simple softmax is not the best parameterization for the variational distribution of the latent propagation variable.

Performance on heterophily graphs. Besides the previously

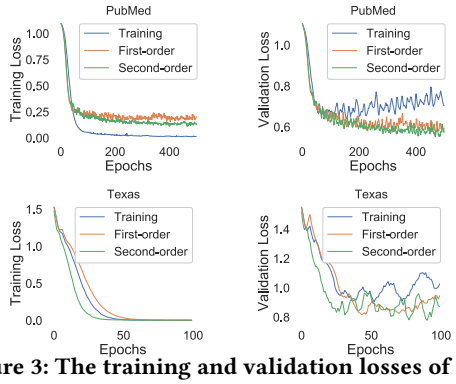

Figure 3: The training and validation losses of L2Q.

Table 3: Semi-supervised classification accuracy (%) on different connections using GCN as the backbone.

Dataset	GCN	Res	JK	Incep	L2S	L2Q
Cora	81.3	78.8	81.1	81.7	**82.6**	81.5
CiteSeer	71.1	70.5	69.8	70.2	71.3	**71.9**
PubMed	78.8	78.6	78.1	77.9	79.4	**79.6**
Actor	30.3	31.3	34.2	32.4	35.0	**35.1**
Cornell	57.0	60.2	64.6	66.5	70.2	**70.5**
Texas	59.5	65.7	66.5	75.6	80.3	**80.5**
Wisconsin	59.9	71.2	74.3	75.1	80.0	**80.1**

mentioned baselines, we also compare our methods with three variants of Geom-GCN [36]: Geom-GCN-I, Geom-GCN-P, and Geom-GCN-S. Table 2 reports the results. (1) We can observe that L2S and L2Q outperform the APPNP backbone on four heterophily graphs, which indicates our framework can still work well on the heterophily graphs. (2) L2S and L2Q consistently improve GCNII* by a large margin and achieve new state-of-the-art results on four heterophily graphs. (3) We can find that the improvement on heterophily graphs is usually larger than that on homophily graphs (Table 1). This is because the neighborhood information is noisy, aggregating the neighborhood information may result in worse performance for GCNII*. In contrast, our L2S and L2Q adaptively learn the process of propagation which can avoid utilizing the structure information which maybe not helpful for heterophily graphs.

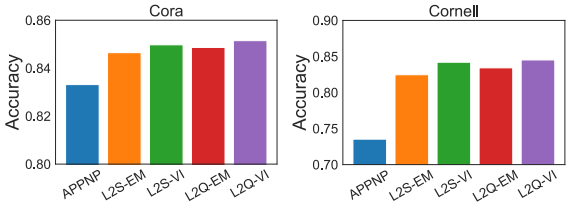

Figure 4: Comparison of different learning algorithms.

Performance with the other backbone. To further show the effectiveness of our framework, we use the GCN as the backbone and compare our methods with the following connection designs which toward alleviate the over-smoothing or capture higher-order information: Residual (Res) [25, 27, 37], Jumping knowledge (JK) [45], and Inception (Incep) [37] connections. Table 3 shows the performance of different connections on homophily and heterophily

Table 4: Classification accuracy (%) results with different predefined propagation steps on Cora.

Method	Propagation Steps					
	2	4	8	16	32	64
GCN	81.1	80.4	69.5	64.9	60.3	28.7
GCN(Drop)	**82.8**	82.0	75.8	75.7	62.5	49.5
JKNet	-	80.2	80.7	80.2	81.1	71.5
JKNet(Drop)	-	**83.3**	82.6	83.0	82.5	83.2
Incep	-	77.6	76.5	81.7	81.7	80.0
Incep(Drop)	-	82.9	82.5	83.1	83.1	83.5
GCNII*	80.2	82.3	82.8	83.5	84.9	**85.3**
L2S	82.2	82.8	**84.9**	84.6	84.6	84.6
L2Q	82.2	83.2	84.8	**84.8**	**85.2**	85.2

graphs. From Table 3, we have the following findings: (1) Our L2S and L2Q outperform the baselines, especially in heterophily with GCN backbone, which suggests that our framework is agnostic to backbones and graphs. (3) Although the advanced connections such as Res and JK can alleviate the over-smoothing, they still do not outperform 2-layer GCN on homophily graphs. Our L2S and L2Q are the only two methods that perform better than GCN across all the datasets. These findings show that our L2P framework can effectively adapt to both heterophily and homophily graphs.

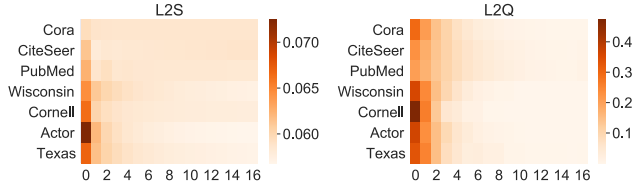

Figure 5: The propagation distributions on different graphs.

6.3 RQ2. Over-smoothing

To answer RQ2, we study how the proposed methods perform as the number of layers increases compared to state-of-the-art (deep) GNNs. We vary the number of layers as {2, 4, 8, 16, 32, 64}. We only report the performance on Cora as we have similar observations on other datasets. Table 4 summaries the semi-supervised results for the deep models with various propagation steps. We observe that the performance of proposed methods consistently improves as increasing the number of layers, which indicates the effectiveness of our L2P framework. For all cases, the proposed methods achieve the best accuracy under a depth beyond 2, which again verifies the

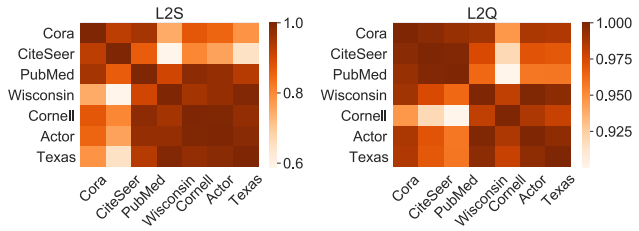

Figure 6: Graph correlation obtained from our learned propagation distributions. Similar graphs are more correlated, such as Cora is closer to CiteSeer than Texas.

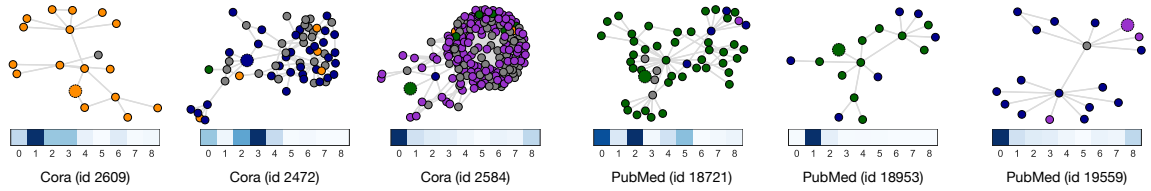

Figure 7: Case studies of the personalized propagation on two homophily datasets. The bigger node in each sub-graph is the test node. The propagation distributions learned by L2Q of the test nodes are visualized with heatmaps (bottom).

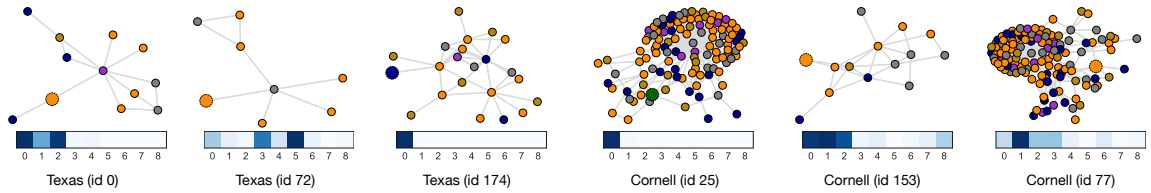

Figure 8: Case studies of the personalized propagation on two heterophily datasets. The bigger node in each sub-graph is the test node. The propagation distributions learned by L2Q of the test nodes are visualized with heatmaps (bottom).

impact of L2P on formulating graph neural networks. Notably, our methods achieve the best performance as we increase the network depth to 64 and the results of our methods remain stable with stacking many layers. On the other hand, the performance of GCN with DropEdge and JKNet drops rapidly as the number of layers exceeds 32, which represents that they suffer from over-smoothing. This phenomenon suggests that with an adaptive and personalized message propagation strategy, L2P can effectively resolve the over-smoothing problem and achieve better performance.

6.4 RQ3. The Effect of Learning Algorithms

To answer RQ3, We first compare the performance of alternate expectation maximization (EM) and iterative variational inference (VI). From Figure 4, we can find that our methods with two learning algorithms both achieve better performance compared to the best results of APPNP, which verifies the effectiveness of our learning algorithm. In general, the iterative VI achieves better performance than the EM algorithm. We then analyze the model loss of stochastic bi-level variational inference with *training* (we optimize ϕ simultaneously with θ on training data without validation), *first-order* and *second-order* approximates. Figures 3 show the learning curves of training loss and validation loss on the Texas and PubMed datasets of L2Q. We can observe that the *training* gets stuck in the overfitting issue attaining low training loss but high validation loss. The gap between training and validation losses is much smaller for first-order and second-order. This demonstrates that the bilevel optimization can significantly improve generalization capability and the first-order approximate is sufficient to prevent the overfitting.

6.5 RQ4. Adaptive Propagation Strategies.

One of the most important properties of our framework is that the learned propagation strategy is interpretable and is different for different types of graphs and nodes. Thus, in this subsection, we investigate if the proposed framework can learn adaptive propagation strategies, which aims to answer RQ4. We visualize the average propagation distribution (via averaging propagation distributions of all nodes) for seven graphs learned by L2S and L2Q with K=16 in Figure 5. The darkness of a step represents the probability that the step is selected for propagation. From Figure 5, we can find that (1) different types of graphs exhibit different propagation distributions although the pre-defined step is 16 for all of them. For instance, the 0-th step probability in heterophily graphs is much larger than that of homophily graphs. This is because that the feature information in those heterophily graphs is much more important than the structure information. (2) The propagation distribution learned by L2Q is much sparse, and the layers on the tail are less likely to be sampled. In Figure 6, we also provide the correlation, i.e. the cosine similarity of learned propagation distributions of different graphs. We clearly observe the correlations between the same types of graphs are large while the correlation between homophily and heterophily graphs is small, which meets our expectation that similar types of graphs should generally have similar propagation strategies.

6.6 RQ5. Personalized Propagation Strategies

To evaluate if L2P can learn good personalized and interpretable propagation for RQ5, we study the propagation strategy for individual nodes. Figures 7 and 8 show the case studies of personalized propagation on homophily and heterophily graphs. In Figures 7 and 8, we plot the 3-hop neighborhood of each test node and use different colors to indicate different labels. We find that a test node with more same class neighbors tends to propagate few steps. In contrast, a test node with fewer same class nodes will probably have more propagation steps to predict truly its label. This observation matches our intuition that different nodes need different propagation strategies, and the prediction of a node will be confused if it has too many propagation steps and thus can not benefit much from message propagation. Additionally, we can find that our framework successfully identifies the propagation steps that are important for predicting the class of nodes on both homophily and heterophily graphs and has a more interpretable prediction process.

7 CONCLUSION

In this paper, we study the problem of learning the propagation strategy in GNNs. We propose learning to propagate (L2P), a general framework to address this problem. Specifically, we introduce the optimal propagation steps as latent variables to help find the maximum-likelihood estimation of the GNN parameters and infer the optimal propagation step for each node via the VEM. Furthermore, we propose L2S and L2Q, two instances to parameterize the variational propagation distribution and frame the variational inference process as a bi-level optimization problem to alleviate the over-fitting problem. Extensive experiments demonstrate that our L2P can achieve state-of-the-art performance on seven benchmark datasets and adaptively capture the personalized and interpretable propagation strategies of different nodes and various graphs.

ACKNOWLEDGMENTS

The authors would like to thank the Westlake University and Bright Dream Robotics Joint Institute for the funding support. Suhang Wang is supported by the National Science Foundation under grant number IIS-1909702, IIS-1955851, and Army Research Office (ARO) under grant number W911NF-21-1-0198.

REFERENCES

[1] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. 2018. Online Learning Rate Adaptation with Hypergradient Descent. In *ICLR*.
[2] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2018. Automatic differentiation in machine learning: a survey. *JMLR* (2018).
[3] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational inference: A review for statisticians. *Journal of the American statistical Association* (2017).
[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and deep locally connected networks on graphs. In *ICLR*.
[5] 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. In *AAAI*.
[6] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.
[7] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *ICML*.
[8] Zhengyu Chen, JiXie Ge, Heshen Zhan, Siteng Huang, and Donglin Wang. 2021. Pareto Self-Supervised Training for Few-Shot Learning. In *CVPR*.
[9] Zhengyu Chen and Donglin Wang. 2021. Multi-Initialization Meta-Learning with Domain Adaptation. In *ICASSP*.
[10] Zhengyu Chen, Ziqing Xu, and Donglin Wang. 2021. Deep transfer tensor decomposition with orthogonal constraint for recommender systems. In *AAAI*.
[11] Enyan Dai and Suhang Wang. 2021. Say No to the Discrimination: Learning Fair Graph Neural Networks with Limited Sensitive Attribute Information. In *WSDM*.
[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*.
[13] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. 2020. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. *NeurIPS* 33 (2020).
[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.
[15] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. 2018. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*.
[16] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. 2019. Learning discrete structures for graph neural networks. In *ICML*.
[17] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2019. Graphnas: Graph neural architecture search with reinforcement learning. *arXiv* (2019).
[18] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.
[19] Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. 1995. The" wake-sleep" algorithm for unsupervised neural networks. *Science* (1995).
[20] Dasol Hwang, Jinyoung Park, Sunyoung Kwon, KyungMin Kim, Jung-Woo Ha, and Hyunwoo J Kim. 2020. Self-supervised Auxiliary Learning with Meta-paths for Heterogeneous Graphs. In *NeurIPS*. 10294–10305.
[21] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv* (2016).
[22] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. 2021. Node Similarity Preserving Graph Convolutional Networks. In *WSDM*.
[23] Mohammad Khan, Shakir Mohamed, Benjamin Marlin, and Kevin Murphy. 2012. A stick-breaking likelihood for categorical data analysis with latent Gaussian models. In *AISTATS*.
[24] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv* (2013).
[25] Thomas N. Kipf and Max Welling. [n.d.]. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[26] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.
[27] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *ICCV*.
[28] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.
[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
[30] Zemin Liu, Wentao Zhang, Yuan Fang, Xinming Zhang, and Steven CH Hoi. 2020. Towards locality-aware meta-learning of tail node embeddings on networks. In *CIKM*.
[31] Dougal Maclaurin, David Duvenaud, and Ryan Adams. 2015. Gradient-based hyperparameter optimization through reversible learning. In *ICML*.
[32] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv* (2016).
[33] Shakir Mohamed and Danilo J Rezende. 2015. Variational information maximisation for intrinsically motivated reinforcement learning. In *NIPS*.
[34] Mark EJ Newman. 2002. Assortative mixing in networks. *Physical review letters* (2002).
[35] Kenta Oono and Taiji Suzuki. 2019. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *ICLR*.
[36] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2019. Geom-GCN: Geometric Graph Convolutional Networks. In *ICLR*.
[37] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *ICLR*.
[38] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Yiqi Wang, Jiliang Tang, Charu C. Aggarwal, Prasenjit Mitra, and Suhang Wang. 2020. Investigating and Mitigating Degree-Related Biases in Graph Convoltuional Networks. In *CIKM*.
[39] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[40] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *ICML*.
[41] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE TNNLS* (2020).
[42] Teng Xiao, Shangsong Liang, and Zaiqiao Meng. 2019. Hierarchical neural variational model for personalized sequential recommendation. In *WWW*.
[43] Teng Xiao, Shangsong Liang, Weizhou Shen, and Zaiqiao Meng. 2019. Bayesian deep collaborative matrix factorization. In *AAAI*.
[44] Teng Xiao and Donglin Wang. 2021. A general offline reinforcement learning framework for interactive recommendation. In *AAAI*.
[45] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*.
[46] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *ICML*.
[47] Lingxiao Zhao and Leman Akoglu. 2019. PairNorm: Tackling Oversmoothing in GNNs. In *ICLR*.
[48] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2021. GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks. In *WSDM*.
[49] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. 2018. On Learning Intrinsic Rewards for Policy Gradient Methods. *NIPS* (2018).
[50] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. 2020. Towards Deeper Graph Neural Networks with Differentiable Group Normalization. In *NeurIPS*.
[51] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. 2019. Auto-gnn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184* (2019).
[52] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. *NeurIPS* (2020).
[53] Daniel Zügner and Stephan Günnemann. 2018. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR*.

A APPENDIX

A.1 Derivations of the Evidence Lower Bound

$$\begin{aligned}
&\log p_\theta(\mathbf{y}_n|\mathbf{X},\mathbf{A}) = \log \sum_{t_n=0}^{K} p_\theta(\mathbf{y}_n|GNN(\mathbf{X},\mathbf{A},t_n))\, p(t_n) \\
&= \mathbb{E}_{q(t_n)}[\log p_\theta(\mathbf{y}_n|GNN(\mathbf{X},\mathbf{A},t_n))] - \mathrm{KL}(q(t_n)||p(t_n)) \\
&+ \mathrm{KL}(q(t_n)||p(t_n|\mathbf{X},\mathbf{A},y_n)) \\
&\geq \mathcal{L}(\theta,q) = \mathbb{E}_{q(t_n)}[\log p_\theta(\mathbf{y}_n|\mathbf{X},\mathbf{A},t_n)] - \mathrm{KL}(q(t_n)||p(t_n)). \qquad (18)
\end{aligned}$$

The inequality holds since the $KL(q(t_n)||p(t_n|\mathbf{X},\mathbf{A},y_n))$ is always no less than zero. The ELBO itself is a lower bound on the log evidence (the log-likelihood), whilst the variational distribution $q(t_n)$ serves as an approximation of the posterior $p(t_n|\mathbf{X},\mathbf{A},y_n)$ [3].

A.2 Derivations of the Bi-level Gradient

$\theta^{(i)}$ is a function of ϕ due to Eq. (15), we can directly back-propagate the gradient through $\theta^{(i)}$ to ϕ. Based on the chain rule, the gradient $\nabla_\phi \mathcal{L}_{\text{val}}(\theta^{(i)}, \phi^{(i-1)})$ can be approximated as follows:

$$\begin{aligned}
&\nabla_\phi \mathcal{L}_{\text{val}}(\theta^{(i)}, \phi^{(i-1)}) \qquad (19) \\
&= \nabla_\phi \mathcal{L}_{\text{val}}(\bar{\theta}^{(i)}, \phi^{(i-1)}) + \nabla_\phi \mathcal{L}_{\text{val}}(\theta^{(i)}, \bar{\phi}^{(i-1)}) \\
&= \nabla_\phi \mathcal{L}_{\text{val}}(\bar{\theta}^{(i)}, \phi^{(i-1)}) + \nabla_{\theta^{(i)}} \mathcal{L}_{\text{val}}(\theta^{(i)}, \bar{\phi}^{(i-1)}) \nabla_\phi \theta^{(i)}(\phi) \\
&= \nabla_\phi \mathcal{L}_{\text{val}}(\bar{\theta}^{(i)}, \phi^{(i-1)}) + \\
&\nabla_{\theta^{(i)}} \mathcal{L}_{\text{val}}(\theta^{(i)}, \bar{\phi}^{(i-1)}) \nabla_\phi (\theta^{(i-1)} - \eta_\theta \nabla_\theta \mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)})) = \\
&\nabla_\phi \mathcal{L}_{\text{val}}(\bar{\theta}^{(i)}, \phi^{(i-1)}) - \eta_\theta \nabla_\theta \mathcal{L}_{\text{val}}(\theta^{(i)}, \bar{\phi}^{(i-1)}) \nabla_\phi \nabla_\theta \mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)}).
\end{aligned}$$

In the last line, we make a Markov assumption that $\nabla_\phi \theta^{i-1} \approx 0$, assuming that at iteration step i, given θ_{i-1} we do not care about how the values of ϕ from previous steps led to θ_{i-1}. This assumption can decrease the computation cost, and it has already shown empirical success in prior works on the bi-level optimization [1, 2]. For the second-order term $\nabla_\theta \mathcal{L}_{\text{val}}(\theta^{(i)}, \bar{\phi}^{(i-1)}) \nabla_\phi \nabla_\theta \mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)})$, we further propose an efficient approximation of it by utilizing the first-order Taylor expansion of $\nabla_\theta \nabla_\phi \mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)})$. Specifically, for any vector $v \in \mathbb{R}^{|\theta|}$, with small $\epsilon > 0$, we have:

$$\begin{aligned}
&v^\top \cdot \nabla_\theta \nabla_\phi \mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)}) \qquad (20) \\
&\approx \frac{1}{\epsilon}(\nabla_\phi \mathcal{L}_{\text{train}}(\theta^{(i-1)} + \epsilon v, \phi^{(i-1)}) - \nabla_\phi \mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)})).
\end{aligned}$$

Given this, $\nabla_\phi \mathcal{L}_{\text{val}}(\theta^{(i)}, \phi^{(i-1)})$ in Eq. (19) can be approximated as:

$$\begin{aligned}
&\nabla_\phi \mathcal{L}_{\text{val}}(\bar{\theta}^{(i)}, \phi^{(i-1)}) - \eta_\theta \nabla_\theta \mathcal{L}_{\text{val}}(\theta^{(i)}, \phi^{(i-1)}) \cdot \\
&\nabla_\phi \nabla_\theta \mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)}) = \nabla_\phi \mathcal{L}_{\text{val}}(\bar{\theta}^{(i)}, \phi^{(i-1)}) \qquad (21) \\
&- \eta_\theta \frac{1}{\epsilon}(\nabla_\phi \mathcal{L}_{\text{train}}(\theta^{(i-1)} + \epsilon v, \phi^{(i-1)}) - \nabla_\phi \mathcal{L}_{\text{train}}(\theta^{(i-1)}, \phi^{(i-1)})),
\end{aligned}$$

where $v = \nabla_\theta \mathcal{L}_{\text{val}}(\theta^{(i)}, \bar{\phi}^{(i-1)})$.

A.3 Datasets Description and Statistics

In our experiments, we use the following real-world datasets. The statistics of datasets are given in Table 5.

Cora, PubMed and CiteSeer are citation and homophily graphs, which are among the most widely used benchmarks for semi-supervised node classification [25, 39, 46]. In these citation datasets, nodes are documents, and edges are citations. Each node is assigned a class label based on the research field. These datasets use a bag of words representation as the feature vector for each node.

Table 5: The statistics of datasets.

Dataset	Classes	Nodes	Edges	Features
Cora	7	2,708	5,429	1,433
Citeseer	6	3,327	4,732	3,703
PubMed	3	19,717	44,338	500
Actor	5	7,600	26,659	932
Texas	5	183	309	1,703
Cornell	5	183	295	1,703
Wisconsin	5	251	499	1,703

Actor is a heterophily graph representing actor co-occurrence in Wiki pages [36] based on the film-director-actor-writer network.
Texas, Wisconsin and Cornell are heterophily graphs representing links between web pages of the corresponding universities, originally collected by the CMU WebKB project. We use the pre-processed datasets in [36]. These datasets are web networks, where nodes and edges represent web pages and hyperlinks, respectively.

A.4 Baselines and Implementations

GCN: GCN [25] is a widely used graph convolutional model.
GAT: GAT [39] utilizes the attention mechanism and assigns different weights to different neighborhoods in the propagation step.
JK-Net: JK-Net [45] utilizes dense connections to leverage different neighbor ranges for better representations of nodes.
APPNP: APPNP [26] adds the original node feature to the representation learned by each layer, which can well preserve the personalized information of nodes so as to alleviate over-smoothing.
DAGNN: DAGNN [50] proposes adaptive weighting to integrate representations from different aggregation steps into the last layer.
IncepGCN: IncepGCN [37] utilizes inception backbones with graph convolution layers to capture the information from different hops.
GCNII: GCNII [7] improves GCN by adding residual connection and identity mapping. We compare our methods with GCNII* which is a variant of GCNII employing two weight matrices, since it can generally achieve better performance than GCNII as shown in [7].
Implementations. For all baselines, we used the official implementation publicly released by the authors on Github.
Hardware. We ran our experiments on GeForce RTX 2080 Ti (11G).

A.5 Experimental setup

Dataset splitting. For homophily graphs (Cora, PubMed, and CiteSeer), we follow the widely used semi-supervised setting in [25, 39, 46] and apply the standard fixed training/validation/testing split with 20 nodes per class for training, 500 nodes for validation and 1,000 nodes for testing. For heterophily graphs, we use the feature vectors, class labels, and 10 random splits (48%/32%/20% of nodes per class for train/validation/test) from [36, 52].
Parameter setting. We randomly initialize the parameters. For our methods, the hyper-parameter search spaces are as follows: dropout (0.2, 0.4, 0.6), learning rate (0.001, 0.005, 0.01), hidden layer size (64, 128), L2 weight-decay (5e-4, 1e-4, 5e-6, 1e-6). For all methods. the propagation steps K is tuned from $(2, 4, \ldots, 32, 64)$. For each search of hyper-parameter configuration, we run the experiments with 20 random seeds and select the best configuration of hyper-parameters based on average accuracy on the validation set.