A Collective Learning Framework to Boost GNN Expressiveness

Mengyue Hang
Department of Computer Science
Purdue University
hangm@purdue.edu

Jennifer Neville
Department of Computer Science
Purdue University
neville@cs.purdue.edu

Bruno Ribeiro
Department of Computer Science
Purdue University
ribeiro@cs.purdue.edu

Abstract

Graph Neural Networks (GNNs) have recently been used for node and graph classification tasks with great success. Unfortunately, existing GNNs are not universal (i.e., most-expressive) graph representations. In this work, we propose *collective learning*, a general collective classification Monte Carlo approach for graph representation learning that provably increases the representation power of existing GNNs. We show that our use of Monte Carlo sampling is key in these results. Our experiments consider the task of inductive node classification across partially-labeled graphs using five real-world network datasets and demonstrate a consistent, significant boost in node classification accuracy when our framework is combined with a variety of state-of-the-art GNNs.

1 Introduction

Graph Neural Networks (GNNs) have recently shown great success at node and graph classification tasks [7, 11, 15, 35]. GNNs have been applied in both transductive settings (where the test nodes are embedded in the training graph) and inductive settings (where the training and test graphs are disjoint). Despite their success, existing GNNs are no more powerful than the Weisfeiler-Lehman (WL) graph isomorphism test, and thus, inherit its shortcomings, i.e. they are not universal (most-expressive) graph representations [2, 19, 20, 35]. In other words, these GNNs (which we refer to as WL-GNNs and also include GCNs [11]) are not expressive enough for some node classification tasks, since their representation can provably fail to distinguish non-isomorphic nodes with different labels.

At the same time, a large body of work in relational learning has focused on strengthening poorly-expressive (i.e., local) classifiers in relational models (e.g., relational logistic regression, naive Bayes, decision trees [22]) in *collective classification* frameworks, by incorporating dependencies among node labels and propagating inferences during classification to improve performance, particularly in semi-supervised settings [12, 25, 34].

In this work, we theoretically and empirically investigate the hypothesis that, by explicitly incorporating label dependencies among neighboring nodes via predicted label sampling—akin to how collective classification improves not-so-expressive classifiers—it is possible to devise an add-on training and inference procedure that can improve the expressiveness of any existing WL-GNN for inductive node classification tasks, which we denote *collective learning*.

Preprint. Under review.

Contributions. We first show that collective classification is provably unnecessary if one can obtain GNNs that are most-expressive. Then, because current WL-GNNs are *not* most-expressive, we propose an add-on general collective learning framework to GNNs that provably boosts their expressiveness, beyond that of an *optimal* WL-GNN[*]. Our framework, which we call *CL-GNN*, involves the use of self-supervised learning and sampled embeddings to incorporate node labels during inductive learning—and it can be implemented with *any* component GNN. In addition to being strictly more expressive than optimal WL-GNNs, we also show how collective learning improves finite d-layer WL-GNNs in practice by extending their power to distinguish non-isomorphic nodes from $d-$hop neighborhoods to $2d$. We also show that, in contrast to our proposed framework, attempts to incorporate collective classification ideas into WL-GNNs without sampled embeddings (e.g., [5, 28, 33]) cannot increase expressivity beyond that of an optimal WL-GNN. Our empirical evaluation shows *CL-GNN* achieves a consistent improvement of node classification accuracy, across a variety of state-of-the-art WL-GNNs, for tasks involving unlabeled and partially-labeled test graphs.

2 Problem formulation

We consider the problem of *inductive* node classification across partially-labeled graphs, which takes as input a graph $G^{(\text{tr})} = (V^{(\text{tr})}, E^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})})$ for training, where $V^{(\text{tr})}$ is a set of $n^{(\text{tr})}$ vertices, $E^{(\text{tr})} \subset V^{(\text{tr})} \times V^{(\text{tr})}$ is a set of edges with adjacency matrix $\boldsymbol{A}^{(\text{tr})}$, $\boldsymbol{X}^{(\text{tr})}$ is a $n^{(\text{tr})} \times p$ matrix containing node attributes as p-dimensional vectors, and $\boldsymbol{Y}_L^{(\text{tr})}$ is a set of observed labels (with C classes) of a connected set of nodes $V_L^{(\text{tr})} \subset V^{(\text{tr})}$, where $V_L^{(\text{tr})}$ is assumed to be a proper subset of $V^{(\text{tr})}$, noting that $V_L^{(\text{tr})} \neq \emptyset$. Let $\boldsymbol{Y}_U^{(\text{tr})}$ be the unknown labels of nodes $V_U^{(\text{tr})} = V^{(\text{tr})} \setminus V_L^{(\text{tr})}$. The goal is to learn a *joint* model of $\boldsymbol{Y}_U^{(\text{tr})} \sim P(\boldsymbol{Y}_U | G^{(\text{tr})})$ and apply this same model to predict hidden labels $\boldsymbol{Y}_U^{(\text{te})}$ in another test graph $G^{(\text{te})}$, i.e., $\hat{\boldsymbol{Y}}_U^{(\text{te})} = \arg\max_{\boldsymbol{Y}_U} P(\boldsymbol{Y}_U | G^{(\text{te})})$. The test graph $G^{(\text{te})}$ can be partially labeled or unlabeled so $V_L^{(\text{te})} \supseteq \emptyset$.

Graph Neural Networks (GNNs), which aggregate node attribute information to produce node representations, have been successfully used for this task. At the same time, relational machine learning (RML) methods, which use collective inference to boost the performance of local node classifiers via (predicted) label dependencies, have also been successfully applied to this task.

Since state-of-the-art GNNs are not most-expressive [19, 20, 35], collective classification ideas may help to improve the expressiveness of GNNs. In particular, collective inference methods often *sample* predicted labels (conditioned on observed labels) to improve the local representation around nodes and approximate the joint distribution $P(\boldsymbol{Y}_U | G^{(\text{te})})$. We also know from recent research that sampling randomized features can boost GNN expressiveness [20]. This leads to the key conjecture of this work Hypothesis 1, which we prove theoretically in Section 4 and validate empirically by extensive experimentation in Section 5.

Hypothesis 1. *Since current Graph Neural Networks (e.g. GCN, GraphSAGE, TK-GNN) cannot produce most expressive graph representations, collective learning (which takes label dependencies into account via Monte Carlo sampling) can improve the accuracy of node classification by producing a more expressive graph representation.*

Why? Because WL-GNNs can extract more information about local neighborhood dependencies via sampling [20], and sampling predicted labels allows the GNN to pay attention to the relationship between node attributes, the graph topology, and label dependencies in local neighborhoods. With *collective learning*, GNNs will be able to incorporate more information into the estimated joint label distribution. Next, we describe our *collective learning* framework.

3 Proposed framework: *Collective Learning* to boost GNNs

In this section, we outline *CL-GNN*. It is a general framework to incorporate any GNN, and combines *self-supervised learning approach* and *Monte Carlo embedding sampling* in an *iterative* process to improve inductive learning on partially labeled graphs.

[*]We use the term optimal WL-GNN to refer to the most expressive version of a GNN–one that has the same distinguishing power as a Weisfeiler-Lehman test. Note this is not a universal graph representation.

Specifically, given a partially labeled training graph $G^{(\text{tr})} = (V^{(\text{tr})}, E^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})})$ with adjacency matrix $\boldsymbol{A}^{(\text{tr})}$ and a partially-labeled test graph $G^{(\text{te})} = (V^{(\text{te})}, E^{(\text{te})}, \boldsymbol{X}^{(\text{te})}, \boldsymbol{Y}_L^{(\text{te})})$ with adjacency matrix $\boldsymbol{A}^{(\text{te})}$. The goal of inductive node classification task is to train a joint model on $G^{(\text{tr})}$ to learn $P(\boldsymbol{Y}_U | G^{(\text{tr})})$ and apply it to $G^{(\text{te})}$ by replacing the input graph $G^{(\text{tr})}$ with $G^{(\text{te})}$. Suppose the graphs $G^{(\text{tr})}$ and $G^{(\text{te})}$, we can define $\boldsymbol{Y}_L^{(\text{tr})}$ as a binary (0-1) matrix of dimension $|V^{(\text{tr})}| \times C$, and $\boldsymbol{Y}_L^{(\text{te})}$ of dimension $|V^{(\text{te})}| \times C$, where the rows corresponding to the one-hot encoding of the (available) labels.

(Background) GNN and representation learning. Given a partially labeled graphs $G^{(\text{tr})}$, WL-GNNs generate node representation by propagating feature information throughout the graph. Specifically, $\forall v \in V^{(\text{tr})}$,

$$P(\boldsymbol{Y}_v | \boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}) = \sigma(\mathbf{W}\boldsymbol{Z}_v + \mathbf{b}), \tag{1}$$

where $\boldsymbol{Z}_v = \text{GNN}(\boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}; \Theta)_v$ is the GNN representation of node v, $\sigma(\cdot)$ is the softmax activation, and Θ, $\mathbf{W}$ and $\mathbf{b}$ are model parameters, which are learned by minimizing the cross-entropy loss between true labels $\boldsymbol{Y}_L^{(\text{tr})}$ and the predicted labels.

The collective learning framework. Following Hypothesis 1, we propose Collective Learning GNNs (*CL-GNN*), which includes label information as input to GNNs to produce a more expressive representation. The overall framework follows three steps: (**Step 1**) Include labels in the input using a random mask; (**Step 2**) Sample predicted labels for whatever is masked, use as input to WL-GNN that also takes all node labels as input, and average representations of the WL-GNN over the sampled predicted labels; (**Step 3**) Perform one optimization step by minimizing a negative log-likelihood upper bound (per Proposition 2). Collective learning for WL-GNNs then consists of iterating over Steps 1-3 for $t = 1, \ldots, T$ iterations. Finally, once optimized, we perform inference via Monte Carlo estimates.

Step 1. Random masking and self supervised learning. The input to GNNs is typically the full graph $G^{(\text{tr})}$. If we included the observed labels $\boldsymbol{Y}_L^{(\text{tr})}$ directly in the input, then it would be trivial to learn a model that predicts part of the input. Instead, we either (*scenario test-unlabeled*) mask all label inputs if the test graph $G^{(\text{te})}$ is expected to be unlabeled; or (*scenario test-partial*) if $G^{(\text{te})}$ is expected to have partial labels, we apply a mask to the labels we wish to predict in training so they do not appear in the input $\boldsymbol{Y}_L^{(\text{tr})}$.

In (*scenario test-partial*), where $G^{(\text{te})}$ is expected to have some observed labels, at each stochastic gradient descent step, we randomly sample a binary mask $\boldsymbol{M} \sim \text{Uniform}(\mathcal{M})$ from a set of masks, where $\boldsymbol{M}$ is a $|V^{(\text{tr})}| \times C$ binary (0-1) matrix with the same $|V^{(\text{tr})}|$-dimensional vector in each column. By applying the mask on the observed labels $\boldsymbol{Y}_L^{(\text{tr})}$, the set of true labels is effectively partitioned into two parts, where part of true labels $\boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}$ are used as input to *CL-GNN*, and the other part $\boldsymbol{Y}_L^{(\text{tr})} \odot \overline{\boldsymbol{M}}$ are used as optimization target, where $\overline{\boldsymbol{M}} := \mathbf{1} - \boldsymbol{M}$ is the bitwise negated matrix of $\boldsymbol{M}$.

Step 2. *CL-GNN* loss and its representation averaging. At the t-th step of our optimization —these steps can be coarser than a gradient step— we either (*scenario test-partial*) sample a mask $\boldsymbol{M}^{(t)} \sim \text{Uniform}(\mathcal{M})$ or (*scenario test-unlabeled*) set $\boldsymbol{M}^{(t)} = \mathbf{0}$. For now, we assume we can sample $\hat{\mathbf{Y}}^{(t-1)} = (\hat{\boldsymbol{Y}}_v^{(t-1)})_{v \in V^{(\text{tr})}}$ from an estimate of the distribution $P(\boldsymbol{Y}_v^{(\text{tr})} | \boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}^{(t)}, \boldsymbol{A}^{(\text{tr})})$ —we will come back to this assumption soon. Let $\boldsymbol{X}^{(\text{tr})}_{\boldsymbol{Y}_L^{(\text{tr})}, \hat{\boldsymbol{Y}}^{(t-1)}, \boldsymbol{M}^{(t)}}$ be the matrix concatenation between $\boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}^{(t)} + \hat{\boldsymbol{Y}}^{(t-1)} \odot \overline{\boldsymbol{M}}^{(t)}$ and $\boldsymbol{X}^{(\text{tr})}$, where again $\overline{\boldsymbol{M}} := \mathbf{1} - \boldsymbol{M}$ is the bitwise negated matrix of $\boldsymbol{M}$. Let

$$\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)}; \Theta) = \mathbb{E}_{\hat{\boldsymbol{Y}}^{(t-1)}} \left[\text{GNN}(\boldsymbol{X}^{(\text{tr})}_{\boldsymbol{Y}_L^{(\text{tr})}, \hat{\boldsymbol{Y}}^{(t-1)}, \boldsymbol{M}^{(t)}}, \boldsymbol{A}^{(\text{tr})}; \Theta)_v \right], \tag{2}$$

where GNN represents an arbitrary graph neural network model and Z_v^t is the *CL-GNN*'s representation obtained for node $v \in V^{(\text{tr})}$ at step $t \geq 1$.

Our optimization is defined over the expectation of $\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)})$ w.r.t. to the sampled predicted labels $\hat{\boldsymbol{Y}}^{(t-1)}$ (Equation (2)) and over a loss averaged over all sampled masks (noting that the case where $\boldsymbol{M}^{(t)} = \mathbf{0}$ is trivial):

$$\Theta_t, \mathbf{W}_t, \mathbf{b}_t = \underset{\Theta, \mathbf{W}, \mathbf{b}}{\arg\min}\, \mathbb{E}_{\boldsymbol{M}^{(t)}} \left[- \sum_{v \in V_L^{(\text{tr})}} \overline{\boldsymbol{M}}_v^{(t)} \log \sigma(\mathbf{W}\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)}; \Theta) + \mathbf{b})_{y_v^{(\text{tr})}} \right], \tag{3}$$

where again, $\sigma(\cdot)$ is the softmax activation function, and $V_L^{(\text{tr})}$ are the labeled nodes in training graph.

Obtaining $\hat{\mathbf{Y}}^{(t-1)}$. At iteration t, we use the learned *CL-GNN* model parameter Θ_{t-1} to obtain $\boldsymbol{Z}_v^{(t-1)}$ according to Equation (2) and use the *CL-GNN* model parameters $\mathbf{W}_{t-1}, \mathbf{b}_{t-1}$ to obtain the label prediction recursively

$$\hat{\boldsymbol{Y}}_v^{(t-1)} \sim \text{Categorical}(\sigma(\mathbf{W}_{t-1}\boldsymbol{Z}_v^{(t-1)}(\boldsymbol{M}^{(t)}; \Theta_{t-1}) + \mathbf{b}_{t-1})),\ v \in V^{(\text{tr})}, \tag{4}$$

starting the recursion with $\hat{\boldsymbol{Y}}^{(t-2)} = \mathbf{0}$.

Step 3. Stochastic optimization of Equation (3). Equation (3) is based on a pseudolikelihood, where the joint distribution of the labels $\{\boldsymbol{Y}_v^{(\text{tr})} : v \in V_L^{(\text{tr})} \text{ s.t. } \overline{\boldsymbol{M}}_v^{(t)} = 1\}$ is decomposed as marginal distributions resulting in the sum over $V_L^{(\text{tr})}$ in Equation (3). In order to optimize Equation (3), we compute gradient estimates w.r.t. Θ and $\mathbf{b}$ using the following sampling procedure.

1. We first need to compute an unbiased estimate of $\{\boldsymbol{Z}_v^{(t-1)}\}_{v \in V_L^{(\text{tr})}}$ in Equation (2) using K i.i.d. samples $\hat{\boldsymbol{Y}}^{(t-1)}$ from the model obtained at time step $t-1$ (as describe above). Note that the time/space complexity of the *CL-GNN* is K times the time/space complexity of the corresponding GNN model as we have to compute K representations for each node at each stochastic gradient step.

2. Next, we need an unbiased estimate of the expectation over $\boldsymbol{M}^{(t)}$ in Equation (3). In *(scenario test-partial)* the unbiased estimates are obtained by sampling $\boldsymbol{M}^{(t)} \sim \text{Uniform}(\mathcal{M})$, in the *(scenario test-unlabeled)* the value obtained is exact since $\boldsymbol{M}^{(t)} = \mathbf{0}$. Proposition 2 shows that the above procedure is a proper surrogate upperbound of the loss function

Inference with learned model. We apply the same procedure as in *obtaining* $\hat{\mathbf{Y}}^{(t-1)}$, but transferring the learned parameters to a different test graph. Specifically, at iteration t, *CL-GNN* parameters $\Theta_t, \boldsymbol{W}_t, \boldsymbol{b}_t$ are learned according to Equation (3) on the training graph $G^{(\text{tr})}$. Suppose the iteration contains J gradient steps, thus J sampled masks are used. Given an any-size attributed graph $G^{(te)}$, we sample another J masks $\boldsymbol{M}^{(t)}$ of size $|V^{(\text{te})}|$, either (*scenario test-partial*) sampling $\boldsymbol{M}^{(t)} \sim \text{Uniform}(\mathcal{M})$ or (*scenario test-unlabeled*) set $\boldsymbol{M}^{(t)} = \mathbf{0}$. We also sample K predicted labels $\{\hat{\mathbf{Y}}_1^{(\text{te}),(t-1)}, \cdots, \hat{\mathbf{Y}}_K^{(\text{te}),(t-1)}\}$ from the predicted probability distribution where $\hat{\mathbf{Y}}_k^{(\text{te}),(t-1)} = (\hat{\boldsymbol{Y}}_v^{(t-1)})_{v \in V^{(te)}}$. The node representations for $v \in V^{(\text{te})}$ are obtained using $\boldsymbol{M}^{(t)}$ and $\hat{\mathbf{Y}}_{1,\cdots,K}^{(\text{te}),(t-1)}$:

$$\boldsymbol{Z}_v^{(t)} = \frac{1}{JK}\sum_{j=1}^{J}\sum_{k=1}^{K} \text{GNN}(\boldsymbol{X}^{(\text{te})}_{\boldsymbol{Y}_L^{(\text{te})}, \hat{\boldsymbol{Y}}_k^{(\text{te}),(t-1)}, \boldsymbol{M}_j^{(t)}}, \boldsymbol{A}^{(\text{te})}; \Theta_t)_v,\ \forall v \in V^{(\text{te})}, \tag{5}$$

where again, J is the number of gradient steps per iteration t and K is the number of Monte Carlo samples of $\hat{\mathbf{Y}}^{(\text{te}),(t-1)}$. Then the label predictions are obtained using the learned *CL-GNN* parameters $\boldsymbol{W}_t, \boldsymbol{b}_t$:

$$\hat{\boldsymbol{Y}}_v^{(t)} \sim \text{Categorical}(\sigma(\mathbf{W}_t\boldsymbol{Z}_v^{(t)} + \mathbf{b}_t)_v),\ \forall v \in V^{(\text{te})}.$$

Note that the test label predictions $\hat{\boldsymbol{Y}}^{(\text{te}),(t)}$ are also recursively updated, and the recursion starts with $\hat{\boldsymbol{Y}}^{(\text{te}),(t-2)} = \mathbf{0}$.

4 Collective learning analysis

Is collective classification able to better represent target label distributions than graph representation learning? The answer to this question is both *yes (for WL-GNNs) and no (for most-expressive representations)*. Theorem 1 shows that a most-expressive graph representation [20, 16, 31] would not benefit from a collective learning boost. All proofs can be found in the Appendix.

Theorem 1 (Collective classification can be unnecessary)**.** *Consider the task of predicting node labels when no labels are available in test data. Let $\Gamma^\star(v, G^{(te)})$ be a most-expressive representation of node $v \in V^{(te)}$ in graph $G^{(te)}$. Then, for any collective learning procedure predicting the class label of $v \in V^{(te)}$, there exists a classifier that takes $\Gamma^\star(v, G^{(te)})$ as input and predicts the label of v with equal or higher accuracy.*

While Theorem 1 shows that the most-expressive graph representation does not need collective classification, WL-GNNs are not most-expressive [19, 20, 35]. Indeed, Theorem 2 and Proposition 1 show that *CL-GNN* boosts the expressiveness of optimal WL-GNN and practical WL-GNNs, respectivelly. Then, we show that the stochastic optimization in Step 3 optimizes a loss surrogate upper bound.

4.1 Expressive power of *CL-GNN*

Morris et al. [19] and Xu et al. [35] show that WL-GNNs are no more powerful in distinguishing non-isomorphic graphs and nodes as the standard Weisfeiler-Lehman graph isomorphism test (1-WL or just WL test). Two nodes are assumed isomorphic by the WL test if they have the same color assignment in the stable coloring. The node-expressivity of a parameterized graph representation Γ (with parameter $\Gamma(\cdot; \boldsymbol{W})$) can then be determined by the set of graphs for which Γ can identify non-isomorphic nodes:

$$\mathcal{G}(\Gamma) = \{G : \exists \boldsymbol{W}^\star_G, \text{ s.t. } \forall u, v \in V_G, \Gamma(G; \boldsymbol{W}^\star_G)_v = \Gamma(G; \boldsymbol{W}^\star_G)_u \text{ iff } u, v \text{ are isomorphic}, G \in \mathbb{G}\},$$

where $\mathbb{G}$ is the set of all any-size attributed graphs, V_G is the set of nodes in graph G. We call $\mathcal{G}(\Gamma)$ the *identifiable set* of graph representation Γ.

The *most expressive* graph representation $\Gamma^\star$ has an *identifiable set* of all any-size attributed graphs, i.e. $\mathcal{G}(\Gamma^\star) = \mathbb{G}$. We refer to the WL-GNN that is equally expressive as WL test as the *optimal* WL-GNN (or WLGNN*), which is at least as expressive as all other WL-GNNs.

In this section we show that *collective learning* can boost the optimal WLGNN*, i.e., the identifiable set of WLGNN* is a proper subset of collective learning over WLGNN* (denoted *CL-GNN**)

$$\mathcal{G}(\text{WLGNN}^\star) \subsetneq \mathcal{G}(\textit{CL-GNN}^\star).$$

Theorem 2 (*CL-GNN** expressive power)**.** *Let WLGNN* be an optimal WL-GNN. Then, the collective learning representation of Equation (2), using WLGNN* as the GNN component, (denoted* CL-GNN**) is strictly more expressive than this WLGNN* representation model applied to the same tasks.*

Theorem 2 answers Hypothesis 1, by showing that by incorporating collective learning and sampling procedures, *CL-GNN* can boost the expressiveness of WL-GNNs, including the optimal WLGNN*.

Corollary 1. *Consider a graph representation learning method that, at iteration t, replaces $\hat{\boldsymbol{Y}}^{(t-1)}$, in Equations (2) and (4) with a deterministic function over $\boldsymbol{Z}^{(t-1)}$, e.g., a softmax function that outputs $(P(\hat{\boldsymbol{Y}}_v^{(t-1)}|\boldsymbol{Z}^{(t-1)}))_{v \in V^{(tr)}}$. Then, such method will be no more expressive than the optimal WLGNN* and, hence, less expressive than* CL-GNN*.

Corollary 1 shows that existing graph representation methods that —on the surface— may even look like *CL-GNN*, but do not perform the crucial step of sampling $(\hat{\boldsymbol{Y}}_v^{(t-1)})_{v \in V^{(tr)}}$, unfortunately, are no more expressive than WL-GNNs. Examples of such methods include [5, 18, 28, 33].

Next, we see that the practical benefits of collective learning are even greater when the WL-GNN has limited expressive power due to a constraint on the number of message-passing layers.

4.2 How *CL-GNN* further expands the power of few-layer WL-GNNs

A d-layer ($d > 1$) WL-GNN will only aggregate neighborhood information within d hops of any given node (i.e., over a d-hop egonet, defined as the graph representing the connections among all nodes that are at most d hops away from the center node). In practice —mostly for computational reasons— WL-GNNs have many fewer layers than the graph's diameter D, i.e., $d < D$. For instance, GCN [11] and GraphSAGE [7] both used $d = 2$ in their experiments. Hence, they cannot differentiate two non-isomorphic nodes that are isomorphic within their d-hop neighborhood. We now show that *CL-GNN* can gather $2d$-hop neighborhood information with a d-layer WL-GNN.

Proposition 1. *Let G_v^d be the d-hop egonet of a node v in graph G with diameter $D > d$. Let v_1 and v_2 be two non-isomorphic nodes whose d-hop egonets are isomorphic (i.e., $G_{v_1}^d$ is isomorphic to $G_{v_2}^d$) but $2d$-hop egonets are not isomorphic. Then, a WL-GNN representation with d layers will generate identical representations for v_1 and v_2 while* CL-GNN *is capable of giving distinct node representations.*

Proposition 1 shows that collective learning has yet another benefit: *CL-GNN* further boosts the power of WL-GNNs with limited message-passing layers by gathering neighborhood information within a larger radius. Specifically, *CL-GNN* built on a WL-GNN with d layers can enlarge the effective neighborhood radius from d to $2d$ in Equation (2), while WL-GNN would have to stack $2d$ layers to achieve the same neighborhood radius, which in practice may cause optimization challenges (i.e., $d = 2$ is a common hyperparameter value in the literature).

4.3 Optimization of *CL-GNN*

Proposition 2. *If* $\forall v \in V_L^{(tr)}$, $\nabla_\Theta(\mathbf{W}\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)};\Theta))_{y_v^{(tr)}}$ *is bounded (e.g., via gradient clipping), then the optimization in Equation (3), with the unbiased sampling of* $\{\boldsymbol{Z}_v^{(t-1)}\}_{v\in V^{(tr)}}$ *and* $\boldsymbol{M}^{(t)}$ *described above, results in a Robbins-Monro [29] stochastic optimization algorithm that optimizes a surrogate upper bound of the loss in Equation (3).*

Since the optimization objective in Equation (3) is computationally impractical, as it requires computing all possible binary masks and label predictions, Proposition 2 shows that the sampling procedures used in *CL-GNN* that considers K samples of label predictions and a random mask at each gradient step is a feasible approach of estimating an unbiased upper bound of the objective.

5 Experiments

5.1 Experiment Setup

Datasets. We use datasets of Cora, Pubmed, Friendster, Facebook, and Protein. The largest dataset (Friendster [32]) has 43,880 nodes, which is a social network of users where the node attributes include numerical features (e.g number of photos posted) and categorical features (e.g. gender, college, music interests, etc) encoded as binary one-hot features. The node labels represent one of the five age groups. Please refer to Appendix F for more details.

Train/Test split. To conduct inductive learning tasks, for each dataset we split the graphs for training and testing, and the nodes are sampled to guarantee that there is no overlapping between any two sets of $V_L^{(\text{tr})}$, $V_L^{(\text{te})}$ and V_T. In our experiments, we tested two different label rates in test graph: 0 (unlabeled) and 50%. We run five trials for all the experiments, and in each trial we randomly split the nodes/graphs for training/validation and testing.

As our method can be applied to any GNN models, we tested three GNNs as examples:

- GCN [11] which includes two graph convolutional layers. Here we implemented an inductive variant of the original GCN model for our tasks.
- Supervised GraphSage [7] (denoted by GS) with Mean pooling aggregator. We use sample size of 5 for neighbor sampling.
- Truncated Krylov GCN [15] (denoted by TK), a recent GNN model that leverages multi-scale information in different ways and are scalable in depth. The TK has stronger expressive power and achieved state-of-the-art performance on node classification tasks. We implemented Snowball architecture which achieved comparable performance with the other truncated Krylov architecture according to the original paper.

For each of GNNs, we compare its baseline performance (on its own) to the performance achieved using collective learning in *CL-GNN*. Note that we set the number of layers to be 2 for GCN [11] and GraphSage [7] as set in their original papers, and use 10 layers for TK [15]. For a fair comparison, the baseline GNN and *CL-GNN* are trained with the same hyper-parameters, e.g. number of layers, hidden dimensions, learning rate, early-stopping procedures. Please refer to Appendix F for details.

In addition, we also compare to three relational classifiers, ICA [14], PL-EM [25] and GMNN [28]. The first two models apply collective learning and inference with simple local classifiers Naive Bayes for PL-EM and Logistic regression for ICA. GMNN is the state-of-the-art collective model with GNNs, which uses two GCN models to model label dependency and node attribute dependency respectively. All the three models take true labels in their input, thus we use $\boldsymbol{Y}_L^{(\text{tr})}$ for training and $\boldsymbol{Y}_L^{(\text{te})}$ for testing.

We report the average accuracy score and standard error of five trials for the baseline models, and compute the absolute improvement of accuracy of our method over the corresponding base GNN. We

report the *balanced* accuracy scores on Friendster dataset as the labels are highly imbalanced. To evaluate the significance of our model's improvements, we performed a paired t-test with five trials.

5.2 Results

The node classification accuracy of all the models is shown in Table 1. Our proposed collective learning boost is denoted as +CL (for **C**ollective **L**earning) in the results and our model performance (absolute % of improvement over the corresponding baseline GNN) is shown in shaded area. Numbers in bold represent significant improvement over the baseline GNN based on a paired t-test ($p < 0.05$), and numbers with * is the best performing method in each column.

Comparison with baseline GNN models. Table 1 shows that our method improves the corresponding non-collective GNN models for all the three model architectures (i.e. GCN, GraphSage and TK). Although all the models have large variances over multiple trials — which is because different parts of the graphs are being trained and tested on in different trials, our model consistently improves the baseline GNN. The results from a paired t-test comparing the performance of our method and the corresponding non-collective GNN shows that the improvement is almost always significant (marked as bold), with only four exceptions. Comparing the gains on different datasets, our method achieved smaller gains on Friendster. This is because the Friendster graph is much more sparse than all other graphs (e.g. edge density of Friendster is 1.5e-4 and edge density of Cora is 1.44e-3 [32]), which makes it hard for any model to propagate label information and capture the dependency.

Moreover, comparing the improvement over GCN and TK, we can observe that in general our method adds more gains to GCN performance. For example, with 3% training labels on Cora, our method when combining with GCN has an average of 6.29% improvement over GCN, 2.35% improvement over GraphSage, and 0.96% improvement over TK. This is in line with our assumption Hypothesis 1 that collective learning can help GNNs produce a more expressive representation. As GCN is provably less expressive than TK [15], there is a larger room to increase its expressiveness.

Note the we use different trials for the two test label rates, the gains are generally larger when 50% of the labels are available. For example, when combining with GCN, the improvements of our method are 6.29% and 1.72% for unlabeled Cora and Facebook test sets, but with partially-labeled test data, the improvements are 15.69% and 2.95% respectively. This shows the importance of modeling label dependency especially when the some test data labels are observed.

Comparison with other relational classifiers The two baseline non-GNN relational models, i.e. PL-EM and ICA generally perform worse than the three GNNs, with the only exception on Protein dataset. This could be because the two non-GNN models generally need a larger portion of labeled set to train the weak local classifier, whereas GNNs utilize a neural network architecture as "local classifier", which is better at representation learning by transforming and aggregating node attribute information. However, when the model is trained with a large training set (e.g. with 30% nodes on Protein dataset), modeling the label dependency becomes crucial. At the same time, our method is still able to improve the performance of the corresponding GNNs.

For GMNN, the collective GNN model, it achieved better performance than its non-collective base model, i.e. GCN, and we can see that our model combining with GCN achieved comparable or slightly better performance than GMNN. When combing with other more powerful GNNs, our model can easily out-perform it, e.g. on Cora, Pubmed and Facebook datasets, the TK performs better than GMNN and our method adds extra gains over TK.

We did two ablation studies to investigate the usage of predicted labels (detailed in Appendix H), which showed that (1) adding predicted labels in model input had extra value comparing to using true labels only, and (2) the gain of our framework is from using samples of the predicted labels rather than random one-hot vectors.

Runtime analysis. *CL-GNN* computes K embeddings at each stochastic gradient step, therefore overall, per-gradient step, *CL-GNN* is K slower than its component WL-GNN. Overall, after T iterations of Steps 1-3, *CL-GNN* total runtime increases by $T \times K$ over the original runtime of its component WL-GNN. Our experiments are conducted on a single Nvidia GTX 1080Ti with 11 GB of shared memory. *CL-GNN* built on GCN on the largest dataset (i.e. Friendster with 43K nodes) takes 66.31 minutes for training and inference (the longest time), with a total of $T = 10$ iterations, while the corresponding GCN takes only 1.04 minutes for the same operations. In the same dataset,

		Cora		Pubmed		Friendster		Facebook		Protein	
# train labels:		85 (3.21%)		300 (1.52%)		641 (1.47%)		80 (1.76%)		7607 (30%)	
% labels in $G^{(te)}$:		0%	50%	0%	50%	0%	50%	0%	50%	0%	50%
Random		14.28 (0.00)	14.28 (0.00)	33.33 (0.00)	33.33 (0.00)	20.00 (0.00)	20.00 (0.00)	50.00 (0.00)	50.00 (0.00)	50.00 (0.00)	50.00 (0.00)
GCN [11]	-	45.90 (3.26)	36.38 (1.35)	52.68 (2.36)	54.11 (4.86)	29.34 (0.55)	28.44 (0.56)	65.85 (1.01)	63.13 (2.12)	75.86 (1.11)	77.54 (1.09)
	+ CL	**+6.29 (1.49)**	**+15.69 (3.20)**	+4.48 (2.33)	**+5.62 (1.17)**	**+0.81 (0.10)***	**+0.90 (0.32)**	**+1.72 (0.48)**	**+2.95 (0.84)**	**+1.22 (0.51)**	**+0.75 (0.33)**
GS [7]	-	50.69 (1.50)	48.42 (2.82)	59.34 (3.47)	58.52 (5.42)	28.10 (0.59)	28.10 (0.48)	64.56 (0.92)	62.99 (0.88)	73.85 (1.12)	73.01 (2.28)
	+ CL	**+2.35 (0.56)**	**+4.52 (0.84)**	**+1.48 (0.41)**	**+2.42 (0.27)**	+0.31 (0.15)	+0.73 (0.23)	**+2.38 (0.77)**	**+2.05 (0.04)**	**+0.84 (0.12)**	+1.47 (0.63)
TK [15]	-	63.74 (2.61)	55.68 (2.08)	61.13 (5.03)	63.05 (5.15)	28.89 (0.10)	29.30 (0.15)	67.63 (1.03)	65.80 (1.16)	73.65 (1.69)	78.94 (1.50)
	+ CL	**+0.96 (0.30)***	**+7.18 (1.88)***	**+1.00 (0.21)***	**+1.91 (0.75)***	**+0.55 (0.17)**	**+0.45 (0.08)***	**+0.63 (0.26)***	**+2.37 (0.80)***	**+1.31(0.27)***	**+1.36(0.94)**
PL-EM [25]	-	20.70 (0.05)	20.35 (0.05)	38.05 (4.85)	31.70 (4.78)	23.26 (0.01)	26.30 (0.25)	56.17 (7.42)	54.56 (6.17)	78.46 (1.45)	77.95 (1.56)
ICA [14]	-	26.20 (0.51)	31.17 (3.66)	44.40 (1.92)	33.38 (4.69)	25.14 (0.03)	25.08 (0.17)	47.93 (6.04)	59.39 (3.69)	84.88 (3.35)*	84.39 (4.08)*
GMNN [28]	-	49.05 (1.86)	49.36 (2.22)	58.03 (3.26)	62.16 (4.40)	22.20 (0.07)	28.53 (0.64)	65.82 (1.30)	63.45 (2.15)	76.75 (0.74)	75.96 (0.76)

Table 1: Node classification accuracy with unlabeled and partially-labeled test data. Numbers in bold represent significant improvement in a paired t-test at the $p < 0.05$ level, and numbers with * represent the best performing method in each column.

CL-GNN built on TK takes 132.33 minutes for training and inference, while the corresponding TK takes 1.93 minutes. We give a detailed account of running times on smaller graphs in Appendix F. We note that we spent nearly no time engineering *CL-GNN* for speed or for improving our results. Our interest in this paper lies entirely on the gains of a direct application of *CL-GNN*. We fully expect that further engineering advances can significantly reduce the performance penalty and increase accuracy gains. For instance, parallelism can significantly reduce the time to collect K samples in *CL-GNN*.

6 Related work

On collective learning and neural networks. There has been work on applying deep learning to collective classification. For example, Moore & Neville [18] proposed to use LSTM-based RNNs for classification tasks on graphs. They transform each node and its set of neighbors into an unordered sequence and use an RNN to predict the class label as the output of that sequence. Pham et al. [26] designed a deep learning model for collective classification in multi-relational domains, which learns local and relational features simultaneously to encodes multi-relations between any two instances.

The closest work to ours is Fan & Huang [5], which proposed a recurrent collective classification (RCC) framework, a variant of ICA [14] including dynamic relational features encoding label information. Unlike our framework, this method does not sample labels $\hat{\boldsymbol{Y}}$, opting for an end-to-end training procedure. Vijayan et al. [33] opts for a similar no-sample RCC end-to-end training method as [5], now combining a differentiable graph kernel with an iterative stage. Graph Markov Neural Network (GMNN) [28] is another promising approach that applies statistical relational learning to GNNs. GMNNs model the joint label distribution with a conditional random field trained with the variational EM algorithm. GMNNs are trained by alternating between an E-step and an M-step, and two WL-GCNs are trained for the two steps respectively. These studies represent different ideas for bringing the power of collective classification to neural networks. Unfortunately, Corollary 1 shows that, without sampling $\hat{\boldsymbol{Y}}$, the above methods are still WL-GNNs, and hence, their use of collective classification fails to deliver any increase in expressiveness beyond an optimal WL-GNN (e.g., Xu et al. [35]).

In parallel to our work, Jia & Benson [10] considers regression tasks by modeling the joint GNN residual of a target set $(y - \hat{y})$ as a multivariate Gaussian, defining the loss function as the marginal likelihood only over labeled nodes $\hat{y}_L$. In contrast, by using the more general foundation of collective classification, our framework can seamlessly model both classification and regression tasks, and include model predictions over the entire graph $\hat{\boldsymbol{Y}}$ as *CL-GNN*'s input, thus affecting both the model prediction and the GNN training in inductive node classification tasks.

On self-supervised learning and semi-supervised learning. Self-supervised learning is closely related to semi-supervised learning. In fact, self-supervised learning can be seen as a self-imposed semi-supervised learning task, where part of the input is masked (or transformed) and must be predicted back by the model [4, 24, 13, 17]. Recently, self-supervised learning has been broadly applied to achieve state-of-the-art accuracy in computer vision [8, 6] and natural language processing [3] supervised learning tasks. The use of self-supervised learning in graph representation learning is intimately related to the use of pseudolikelihood to approximate true likelihood functions.

Collective classification for semi-supervised learning tasks. Conventional relational machine learning (RML) developed methods to learn joint models from labeled graphs [14, 21], and applied the learned classifier to *jointly* infer the labels for unseen examples with *collective inference*. When

the goal is to learn and predict within a *partially-labeled* graph, RML methods have considered semi-supervised formulations [12, 34, 25] to model the joint probability distribution:

$$P(\boldsymbol{Y}_U|\boldsymbol{Y}_L, \boldsymbol{X}, \boldsymbol{A}).$$

In this case RML methods use both *collective learning* and *collective inference* procedures for semi-supervised learning.

RML methods typically consider *a Markov assumption* to simplify the above expressions —every node $v_i \in V$ is considered conditionally independent of the rest of the network given its Markov Blanket ($\mathcal{MB}(v_i)$). For undirected graphs, this is often simply the set of the immediate neighbors of v_i. Given the Markov blanket assumption, RML methods typically use a local conditional model (e.g., relational Naive Bayes [23], relational logistic regression [27]) to learn and infer labels within the network.

Semi-supervised RML methods utilizes the *unlabeled data* to make better predictions within the network. Given the estimated values of unlabeled examples, i.e.$P(\boldsymbol{Y}_U^{(\text{tr})}|\boldsymbol{Y}_L^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})})$, the local model parameters can be learned by maximizing the *pseudolikelihood* of the labeled part:

$$O = \sum_{v \in V^{(\text{tr})}} \log P(y_v^{(\text{tr})}|\boldsymbol{Y}_{\mathcal{MB}(v)}^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}). \tag{6}$$

The *key* difference between Equation (6) and the GNNs objective in Equation (1): the RML model is always conditioned on the labels (either true labels $\boldsymbol{Y}_L^{(\text{tr})}$ or estimated labels $\hat{\boldsymbol{Y}}_U^{(\text{tr})}$) even when there are no observed labels in the test data, i.e., even when $\boldsymbol{Y}_L^{(\text{te})} = \emptyset$.

The most common form of semi-supervised RML utilizes expectation maximization (EM) [34, 25], which iteratively relearn the parameters given the expected values of the unlabeled examples. For instance, the PL-EM algorithm [25] optimizes the pseudolikelihood over the entire graph:

E-Step: evaluate $P(\boldsymbol{Y}_U^{(\text{tr})}|\boldsymbol{Y}_L^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}, \Theta_{t-1})$

M-Step: learn Θ_t :

$$\Theta_t = \arg\max_{\Theta} \sum_{\boldsymbol{Y}_U^{(\text{tr})} \in \Gamma_U} P(\boldsymbol{Y}_U^{(\text{tr})}|\boldsymbol{Y}_L^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}, \Theta_{t-1}) \sum_{v \in V^{(\text{tr})}} \log P(y_v^{(\text{tr})}|\boldsymbol{Y}_{\mathcal{MB}(v)}^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}, \Theta).$$

In comparison to semi-supervised RML, our proposed framework *CL-GNN* performs collective learning to strengthen GNN, which is a more powerful and flexible "local" classifier compared to the typical weak local classifier used in RML methods (e.g. relational Naive Bayes). For parameter learning (M-step), *CL-GNN* also optimizes pseudolikelihood, but incorporates Monte Carlo sampling from label prediction $\hat{\boldsymbol{Y}}^{(t-1)}$ instead of directly using the predicted probability of unlabeled nodes $P(\boldsymbol{Y}_U^{(\text{tr})}|\boldsymbol{Y}_L^{(\text{tr})}, \boldsymbol{X}^{(\text{tr})}, \boldsymbol{A}^{(\text{tr})}, \Theta_{t-1})$.

Collective inference. When the model is applied to make predictions on unlabeled nodes (in the E-step), *joint (i.e., collective) inference* methods such as variational inference or Gibbs sampling must be applied in order to use the conditionals from Equation (6). This combines the local conditional probabilities with global inference to estimate the joint distribution over the unlabeled vertices, e.g.,:

$$P(\boldsymbol{Y}_U|\boldsymbol{Y}_L, \boldsymbol{X}) \approx Q(\boldsymbol{Y}_U) = \Pi_{v_i \in V_U} Q_i(y),$$

where each component $Q_i(y)$ is iteratively updated.

Alternatively, a Gibbs sampler iteratively draws a label from the corresponding conditional distributions of the unlabeled vertices:

$$\hat{y}_v \sim P(y_v|\hat{Y}_{\mathcal{MB}(v)}, \boldsymbol{Y}_L, \boldsymbol{X}, A), \ \forall v \in V.$$

In Gibbs sampling, it is the sampling of labels that allow us to sample from the joint distribution, which is what enriches the simple models often used in collective inference.

7 Conclusion

In this work, we answer the question "can collective learning and sampling techniques improve the expressiveness of state-of-the-art GNNs in inductive node classification tasks?" We first show that

with the most expressive GNNs there is no need to do collective learning; however, since we do not have the most expressive models, we present the collective learning framework (exemplified in *CL-GNN*), that can be combined with any existing GNNs to provably improve WL-GNN expressiveness. We considered the inductive node classification tasks across graphs, and showed by extensive empirical study that our collective learning significantly boosts GNNs performance on all the tasks.

8 Potential biases

This work presents an algorithm for node classification that works on arbitrary graphs, including real-world social networks, citation networks, etc.. Any classification algorithm that learns from data runs the risk of producing biased predictions reflective of the training data — our work, which learns an inductive node classifier from training graph examples, is no exception. However, our main focus in this paper is on introducing a general algorithmic framework that can increase the expressivity of existing graph representation learning methods, rather than particular real-world applications.

References

[1] Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[2] Chen, Z., Villar, S., Chen, L., and Bruna, J. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems*, pp. 15868–15876, 2019.

[3] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Doersch, C., Gupta, A., and Efros, A. A. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1422–1430, 2015.

[5] Fan, S. and Huang, B. Recurrent collective classification. *Knowledge and Information Systems*, 60(2):741–755, 2019.

[6] Gidaris, S., Bursuc, A., Komodakis, N., Pérez, P., and Cord, M. Boosting few-shot visual learning with self-supervision. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 8059–8068, 2019.

[7] Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

[8] Hénaff, O. J., Razavi, A., Doersch, C., Eslami, S., and Oord, A. v. d. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019.

[9] Jensen, D., Neville, J., and Gallagher, B. Why collective inference improves relational classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 593–598, 2004.

[10] Jia, J. and Benson, A. Outcome correlation in graph neural network regression, 2020.

[11] Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[12] Koller, D., Friedman, N., Džeroski, S., Sutton, C., McCallum, A., Pfeffer, A., Abbeel, P., Wong, M.-F., Heckerman, D., Meek, C., et al. *Introduction to statistical relational learning*. MIT press, 2007.

[13] Lee, H.-Y., Huang, J.-B., Singh, M., and Yang, M.-H. Unsupervised representation learning by sorting sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 667–676, 2017.

[14] Lu, Q. and Getoor, L. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 496–503, 2003.

[15] Luan, S., Zhao, M., Chang, X.-W., and Precup, D. Break the ceiling: Stronger multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1906.02174*, 2019.

[16] Maron, H., Fetaya, E., Segol, N., and Lipman, Y. On the universality of invariant networks. *arXiv preprint arXiv:1901.09342*, 2019.

[17] Misra, I., Zitnick, C. L., and Hebert, M. Shuffle and learn: unsupervised learning using temporal order verification. In *European Conference on Computer Vision*, pp. 527–544. Springer, 2016.

[18] Moore, J. and Neville, J. Deep collective inference. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[19] Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019.

[20] Murphy, R. L., Srinivasan, B., Rao, V., and Ribeiro, B. Relational pooling for graph representations. *arXiv preprint arXiv:1903.02541*, 2019.

[21] Neville, J. and Jensen, D. Iterative classification in relational data. In *Proc. AAAI-2000 workshop on learning statistical models from relational data*, pp. 13–20, 2000.

[22] Neville, J., Jensen, D., Friedland, L., and Hay, M. Learning relational probability trees. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 625–630, 2003.

[23] Neville, J., Jensen, D., and Gallagher, B. Simple estimators for relational bayesian classifiers. In *Third IEEE International Conference on Data Mining*, pp. 609–612. IEEE, 2003.

[24] Noroozi, M. and Favaro, P. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pp. 69–84. Springer, 2016.

[25] Pfeiffer III, J. J., Neville, J., and Bennett, P. N. Overcoming relational learning biases to accurately predict preferences in large scale networks. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 853–863, 2015.

[26] Pham, T., Tran, T., Phung, D., and Venkatesh, S. Column networks for collective classification. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[27] Popescul, A., Ungar, L. H., Lawrence, S., and Pennock, D. M. Towards structural logistic regression: Combining relational and statistical learning. *Departmental Papers (CIS)*, pp. 134, 2002.

[28] Qu, M., Bengio, Y., and Tang, J. Gmnn: Graph markov neural networks. *arXiv preprint arXiv:1905.06214*, 2019.

[29] Robbins, H. and Monro, S. A stochastic approximation method. *Ann. Math. Statist.*, 22(3): 400–407, 09 1951. doi: 10.1214/aoms/1177729586. URL https://doi.org/10.1214/aoms/1177729586.

[30] Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[31] Srinivasan, B. and Ribeiro, B. On the equivalence between node embeddings and structural graph representations. *arXiv preprint arXiv:1910.00452*, 2019.

[32] Teixeira, L., Jalaian, B., and Ribeiro, B. Are graph neural networks miscalibrated? *arXiv preprint arXiv:1905.02296*, 2019.

[33] Vijayan, P., Chandak, Y., Khapra, M. M., Parthasarathy, S., and Ravindran, B. Hopf: Higher order propagation framework for deep collective classification. *arXiv preprint arXiv:1805.12421*, 2018.

[34] Xiang, R. and Neville, J. Pseudolikelihood em for within-network relational learning. In *2008 Eighth IEEE International Conference on Data Mining*, pp. 1103–1108. IEEE, 2008.

[35] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[36] Yang, J., Ribeiro, B., and Neville, J. Stochastic gradient descent for relational logistic regression via partial network crawls. *arXiv preprint arXiv:1707.07716*, 2017.

Supplementary Material of collective learning GNN

A Proof of Theorem 1

We restate the theorem for completeness.

Theorem 1 (Collective classification can be unnecessary)**.** *Consider the task of predicting node labels when no labels are available in test data. Let $\Gamma^\star(v, G^{(te)})$ be a most-expressive representation of node $v \in V^{(te)}$ in graph $G^{(te)}$. Then, for any collective learning procedure predicting the class label of $v \in V^{(te)}$, there exists a classifier that takes $\Gamma^\star(v, G^{(te)})$ as input and predicts the label of v with equal or higher accuracy.*

Proof. Let $\hat{Y}(v) = \varphi(\Gamma^\star(v, G))$ be a classifier function that takes the most expressive representation $\Gamma^\star(v, G)$ of node v as input and outputs a predicted class label for v.

Let $\hat{\boldsymbol{Y}}^{(t)}$ be the set of predicted labels at iteration t of collective classification and let Y_v be the true label of node $v \in V$. Then either (1) $Y_v \perp\!\!\!\perp_{\Gamma^\star(v,G),\varphi} \hat{\boldsymbol{Y}}^{(t)}$, or (2) $Y_v \not\perp\!\!\!\perp_{\Gamma^\star(v,G),\varphi} \hat{\boldsymbol{Y}}^{(t)}$.

Case (1): Given the classifier φ and the most expressive representation $\Gamma^\star(v, G)$, the true label of v is independent of the labels predicted with collective classification. In this case, the predicted labels of v's neighbors offer no additional information and, thus, collective classification is unnecessary.

Case (2): In this case, the true label of v is not independent of the predicted labels. By Theorem 1 of Srinivasan & Ribeiro [31], we know that for any random variable H_v attached to node $v \in V$, it must be that $\exists \varphi'$ a measurable function independent of G s.t.

$$H_v \overset{\text{a.s.}}{=} \varphi'(\Gamma^\star(v, G), \epsilon_v),$$

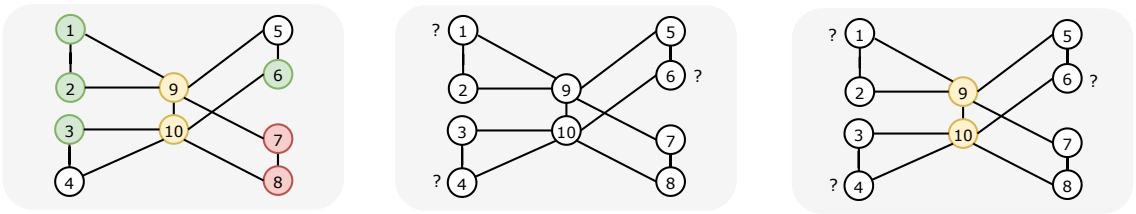
where ϵ_v is an noise source exogenous to G (pure noise), and a.s. implies almost sure equality. Defining $H_v := Y_v$,

$$\varphi'(\Gamma^\star(v, G), \epsilon_v) \not\perp\!\!\!\perp_{\Gamma^\star(v,G),\varphi} \hat{\boldsymbol{Y}}^{(t)},$$

which means $\hat{\boldsymbol{Y}}^{(t)}$ must either be dependent on ϵ_v or contain domain knowledge information about the function φ' that is not in φ. Since $\hat{\boldsymbol{Y}}^{(t)}$ is a vector of random variables fully determined by G and φ, it cannot depend on an exogenous variable ϵ_v, Thus, the predictions must contain domain knowledge of φ'. Hence, we can directly incorporate this domain knowledge into another classifier $\varphi^\dagger$ s.t. $Y_v \perp\!\!\!\perp_{\Gamma^\star(v,G),\varphi^\dagger} \hat{\boldsymbol{Y}}^{(t)}$, for instance $\varphi^\dagger$ is a function of φ'. In this case, $\varphi^\dagger$ will predict the label of v with equal or higher accuracy than collective classification based on predicted labels $\hat{\boldsymbol{Y}}$, which finishes our proof.

□

B Proof of Theorem 2



(a) Training graph (b) unlabeled test graph (c) partially-labeled test graph

Figure 1: Training/testing graphs. Colors represent available node labels, and testing nodes are marked with question marks. WL-GNN cannot differentiate between the red and green nodes.

Theorem 2 (*CL-GNN** expressive power)**.** *Let WLGNN* be an optimal WL-GNN. Then, the collective learning representation of Equation (2), using WLGNN* as the GNN component, (denoted* CL-GNN**) is strictly more expressive than this WLGNN* representation model applied to the same tasks.*

Proof. As defined, WLGNN is a most-expressive WL-GNN. We need to prove $\mathcal{G}(\text{WL-GNN}) \subsetneq \mathcal{G}(\textit{CL-GNN})$. We will do that by first showing $\mathcal{G}(\textit{CL-GNN}) = \mathcal{G}(\text{WL-GNN}) \cup S'$ and then showing that $\exists G \in \mathcal{G}(\textit{CL-GNN})$ s.t. $G \notin \mathcal{G}(\text{WLGNN})$.

$\mathcal{G}(\text{CL-GNN}) = \mathcal{G}(\textit{WL-GNN}) \cup S'$: First, we need to show that for any mask $M \in \mathcal{M}$, $\exists S \subseteq \mathcal{G}(\textit{CL-GNN})$ such that $S = \mathcal{G}(\text{WL-GNN})$. This is clearly true since, for labeled tests, in Equation (2) we can always construct a WLGNN^0 for a *CL-GNN*

$$\boldsymbol{Z}_v^{(t)}(\text{WLGNN}^0) = \mathbb{E}_{\hat{\boldsymbol{Y}}^{(t-1)}}\left[\text{WLGNN}^0(\boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})} \odot M, \hat{\boldsymbol{Y}}^{(t-1)} \odot \overline{M}, \boldsymbol{A}^{(\text{tr})}; \Theta)_v\right], \tag{7}$$

that ignores the $\hat{\boldsymbol{Y}}$ inputs. Similarly, for unlabeled tests, in Equation (2) we can always construct a WLGNN^1 for a *CL-GNN*

$$\boldsymbol{Z}_v^{(t)}(\text{WLGNN}^1) = \mathbb{E}_{\hat{\boldsymbol{Y}}^{(t-1)}}\left[\text{WLGNN}^1(\boldsymbol{X}^{(\text{tr})}, \hat{\boldsymbol{Y}}^{(t-1)}, \boldsymbol{A}^{(\text{tr})}; \Theta)_v\right],$$

that ignores the $\hat{\boldsymbol{Y}}^{(t-1)}$ inputs.

$\exists G \in \mathcal{G}(\text{CL-GNN})$ *s.t.* $G \notin \mathcal{G}(\textit{WL-GNN})$: Let G be the graph in Figure 1. We will first consider the case where the test data has partial labels. The case without labels follows directly from it. Using the graph G in Figure 1(a) (training) and Figure 1(c) (partially-labeled testing) we show that a WL-GNN is unable, in test, to correctly give representations to the left-most nodes $\{1, 2, 3, 4\}$ that are distinct from the right-most nodes $\{5, 6, 7, 8\}$ (the same happens for the unlabeled test graph in Figure 1(b)). We then show that the representation $\boldsymbol{Z}_v^{(t)}$ of Equation (7) is able to distinguish these two sets of nodes.

WL-CNN is not powerful enough to give distinct representations to nodes $\{1, \ldots, 8\}$ *in Figure 1(c)*: Consider giving an arbitrary feature value (say, the "color white") to all uncolored nodes $\{1, \ldots, 8\}$ in Figure 1(c). We will start showing that the 1-WL test is unable to give different colors to the nodes $\{1, \ldots, 8\}$ in this graph. Since WL-GNNs are no more expressive than the 1-WL test [35, 19], showing that the above is a stable coloring for nodes $\{1, \ldots, 8\}$ in the 1-WL test, proves the first part of our result. A stable 1-WL coloring is defined as a coloring scheme on the graph that has a 1-to-1 correspondence with the colors in the previous step of the 1-WL algorithm. Since the input to the hash function of the 1-WL test is the same for all of nodes $v \in \{1, \ldots, 8\}$: The node itself has color white while the color set of the neighbors is the set $\{\text{white}, \text{yellow}\}$. In the next 1-WL round, all the white nodes will be mapped to the same color by the hash function. The colors of node $\{9, 10\}$ will be not the same as $\{1, \ldots, 8\}$. Hence, the initial coloring of all nodes $\{1, \ldots, 8\}$ white and $\{9, 10\}$ yellow is a stable coloring for 1-WL. Consequently, WL-GNN will give the same representation to all nodes in $\{1, \ldots, 8\}$.

CL-GNN gives the same representations within the sets $\{1, \ldots, 4\}$ *and* $\{5, \ldots, 8\}$: At iteration $t \geq 0$ of *CL-GNN*, we start with the base of the recursion $\boldsymbol{Y}^{(t-2)} = \boldsymbol{0}$. Now consider a given mask $\boldsymbol{M}^{(t)} \in \mathcal{M}$. Note that to sample $\hat{\boldsymbol{Y}}_v^{(t-1)}$ for $v \in \{1, \ldots, 8\}$ we apply $\hat{\boldsymbol{Y}}^{(t-2)} = \boldsymbol{0}$ into Equation (2) to obtain $\boldsymbol{Z}_v^{(t-1)}$, and then apply $\boldsymbol{Z}_v^{(t-1)}$ into Equation (4), defining $\boldsymbol{X}^{(\text{tr})}_{\boldsymbol{Y}_L^{(\text{tr})}, \hat{\boldsymbol{Y}}^{(t-1)}, \boldsymbol{M}^{(t)}} = [\boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}^{(t)}, \boldsymbol{0}]$, which will give us $\hat{\boldsymbol{Y}}_v^{(t-1)} \sim P(\boldsymbol{Y}_v^{(t-1)} \,|\, \text{WLGNN}([\boldsymbol{X}^{(\text{tr})}, \boldsymbol{Y}_L^{(\text{tr})} \odot \boldsymbol{M}^{(t)}, \boldsymbol{0}], \boldsymbol{A}^{(\text{tr})}; \Theta)_v)$, and any classes has a non-zero probability of being sampled since our output is a softmax.

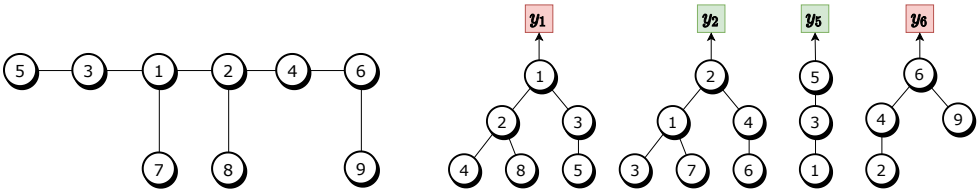
Since nodes $\{1, \ldots, 8\}$ all get the same representation in the above WLGNN, their respective sampled $\hat{\boldsymbol{Y}}_v^{(t-1)}$, $v \in \{1, \ldots, 8\}$, will have the same distribution but possibly not the same values (due to sampling). Note that the nodes $\{1, \ldots, 4\}$ will get the same average in Equation (7) since $\hat{\boldsymbol{Y}}_v^{(t-1)}$, $v \in \{1, \ldots, 4\}$, has the same distribution and the nodes are isomorphic (even given the colors on nodes 9 and 10). Similarly, the nodes $\{5, \ldots, 8\}$ will also get the same average in Equation (7).

CL-GNN gives distinct representations accross the sets $\{1, \ldots, 4\}$ *and* $\{5, \ldots, 8\}$: Finally, we now prove that exists a WL-GNN, which we will denote WLGNN^2, such that $\boldsymbol{Z}_v^{(t)}(\text{WLGNN}^2) \neq \boldsymbol{Z}_u^{(t)}(\text{WLGNN}^2)$ for $v \in \{1, \ldots, 4\}$ and $u \in \{5, \ldots, 8\}$. We will show that there is a joint sample of $\hat{\boldsymbol{Y}}_1^{(t-1)}, \ldots, \hat{\boldsymbol{Y}}_8^{(t-1)}$ where there is no symmetry between the representations of nodes in $\{1, \ldots, 4\}$ and $\{5, \ldots, 8\}$. Since each layer of WLGNN^2 can have different parameters, we can easily encode differences in the number of hops it takes to reach a certain color. Moreover, at any WLGNN^2 layer, the representation of a node can perfectly encode its own last-layer representation and the last-layer representation of its neighbors through a most-expressive multiset representation function [35].

It is enough for us to show that for a sampled $\hat{\boldsymbol{Y}}^{(t-1)}$ the sets of nodes $\{1,\ldots,4\}$ and $\{5,\ldots,8\}$ can get distinct unique representations under WLGNN^2. By unique, we mean, $\{1,\ldots,4\}$ can get representations in WLGNN^2 that cannot be obtained by the nodes in $\{5,\ldots,8\}$. This representation uniqueness makes sure the averages in Equation (2) are different. Without loss of generality we will consider giving a special sampled label to only one node $i \in \{1,5\}$ in one of the sets. The sampled labels $\hat{\boldsymbol{Y}}_i^{(t-1)}$ = green, while all other nodes $\{1,\ldots,8\}\backslash\{i\}$ get red, will happen with non-zero probability, hence, they must be part of the expectation in Equation (2). Note that node 2 (for $i=1$) and 6 (for $i=5$) will feel the effects of the green color in their neighbors differently. That is, for $i=5$ there is a parameter choice for the layers of WLGNN^2 where the representation of node 6 uniquely encodes that the color green is within hops 1 (node 5) and 3 (from node 5 through nodes 9 and 10) of node 6 (if 6 treats its own representation differently from its neighbors). For $i=1$, node 2's representation will encode that green is observed hops 1 (node 1) and 2 (from node 1 through node 9) (similarly, 2 treats its own representation differently from its neighbors). Hence, these representations can be made unique by WLGNN^2, i.e., no other $\hat{\boldsymbol{Y}}^{(t-1)}$ assignments will create the same patterns for nodes 2 and 6, and thus, since WLGNN^2 has most-expressive multiset representations, it can give a unique representation to nodes 2 and 6 for these two unique $\hat{\boldsymbol{Y}}^{(t-1)}$ configurations. These unique representations are enough to ensure $\boldsymbol{Z}_2^{(t)}(\text{WLGNN}^2) \neq \boldsymbol{Z}_6^{(t)}(\text{WLGNN}^2)$ for any $t \geq 1$, which concludes our proof.

□

C Proof of Proposition 1



(a) Training graph (b) 2^{nc}-order neighborhood for label prdiction

Figure 2: WL-GNN using 2nd-order neighborhood cannot differentiate node 1 and 2, but *CL-GNN* built on this WLGNN can break the local isomorphism.

Proposition 1. *Let G_v^d be the d-hop egonet of a node v in graph G with diameter $D > d$. Let v_1 and v_2 be two non-isomorphic nodes whose d-hop egonets are isomorphic (i.e., $G_{v_1}^d$ is isomorphic to $G_{v_2}^d$) but $2d$-hop egonets are not isomorphic. Then, a WL-GNN representation with d layers will generate identical representations for v_1 and v_2 while* CL-GNN *is capable of giving distinct node representations.*

Proof. Let G be the graph in Figure 2(a) with no node features, and let WLGNN be of order 2, meaning it will generate node embeddings based on 2nd-order neighborhoods (shown in (b)). Since node 1 and 2 have the same 2nd-order neighborhood structure, WLGNN will generate identical node representation for them, which gives random label predictions. Meanwhile, as nodes 5 and 6 have distinct 2nd-order neighborhood structures, WLGNN generates different node representations for them, which enables the model to learn from the labels y_5 and y_6. We can assume the predicted label probability $P(\hat{\boldsymbol{Y}}_5^{(0)} = \text{green}) = 0.99$ and $P(\hat{\boldsymbol{Y}}_6^{(0)} = \text{red}) = 0.98$. For *CL-GNN*, at iteration $t=1$, we sample $\hat{\boldsymbol{Y}}^{(0)}$ from the WLGNN output $P(\hat{\boldsymbol{Y}}^{(0)})$ and use the samples as input. In the worst case, nodes $3,4$ and $7,8$ get the same distribution and sampled labels (i.e. $\hat{\boldsymbol{Y}}_3^{(0)} = \hat{\boldsymbol{Y}}_4^{(0)}$, $\hat{\boldsymbol{Y}}_7^{(0)} = \hat{\boldsymbol{Y}}_8^{(0)}$). Since the distribution of $\hat{\boldsymbol{Y}}_5^{(0)}$ and $\hat{\boldsymbol{Y}}_6^{(0)}$ are different, the samples of $\hat{\boldsymbol{Y}}_5^{(0)}$ and $\hat{\boldsymbol{Y}}_6^{(0)}$ are different, which breaks the tie between the 2nd-order neighborhood of nodes 1 and 2. Therefore, *CL-GNN* will produce different node representation starting from iteration $t=1$ for nodes 1 and 2, which enables the model to learn from the training label y_1 and y_2, and thus gives more accurate predictions. □

The advantage of collective inference is more clear when it is used to strengthen less-expressive local classifiers, e.g. logistic regression Although GNN are much powerful than these local classifiers by aggregating high(er)-order graph information, collective learning can still help if GNN fail to make use of "global" information in graphs (or equivalently, if the order of GNN is small than graph diameter). Previous work [9] investigating the power of collective inference also showed that methods for collective inference benefit from *a clever factoring of the space of dependencies*, by arguing that *these (collective inference) methods benefit from information propagated from outside of their local neighborhood. Predictions about the class label on other objects essentially "bundle information" about the graph beyond the immediate neighborhood.*

D Proof of Proposition 2

Proposition 2. *If* $\forall v \in V_L^{(tr)}$, $\nabla_\Theta(\mathbf{W}\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)};\Theta))_{y_v^{(tr)}}$ *is bounded (e.g., via gradient clipping), then the optimization in Equation (3), with the unbiased sampling of* $\{\boldsymbol{Z}_v^{(t-1)}\}_{v\in V^{(tr)}}$ *and* $\boldsymbol{M}^{(t)}$ *described above, results in a Robbins-Monro [29] stochastic optimization algorithm that optimizes a surrogate upper bound of the loss in Equation (3).*

Proof. In our optimization, we only need to sample two variables $\hat{\boldsymbol{Y}}^{(t-1)}$ and $\boldsymbol{M}^{(t)}$. We obtain unbiased bounded-variance estimates of the derivative of the loss function if we sample $\boldsymbol{M}^{(t)} \sim \text{Uniform}(\mathcal{M})$ (and exact values when $\boldsymbol{M}^{(t)} = \mathbf{0}$). We can now compound that with unbiased bounded-variance estimates of the derivative if we estimate the expectation in Equation (2) $\{\boldsymbol{Z}_v^{(t-1)}\}_{v\in V}$ by i.i.d. sampling $\hat{\boldsymbol{Y}}^{(t-1)}$. The loss in Equation (3) is convex on $\boldsymbol{Z}_v^{(t)}$ since the negative log-likelihood of the multi-class logistic regression is convex on $\mathbf{W}$, which means it is also convex on $\boldsymbol{Z}_v^{(t)}$ as the loss is defined on the affine transformation $\mathbf{W}\boldsymbol{Z}_v^{(t)}$. The expectation of the loss always exist, since we assume $\nabla_\Theta(\mathbf{W}\boldsymbol{Z}_v^{(t)}(\boldsymbol{M}^{(t)};\Theta))_{y_v^{(\text{tr})}}$ is bounded for all $\forall v \in V_L^{(\text{tr})}$. Hence, as the loss is convex w.r.t. $\boldsymbol{Z}_v^{(t)}$, the expection w.r.t. $\boldsymbol{Z}_v^{(t)}$ exists, and we obtain an unbiased estimate of $\boldsymbol{Z}_v^{(t)}$, we can apply Jensen's inequality to show that the resulting Robbins-Monro stochastic optimization optimizes an upper bound of the loss in Equation (3). □

E Additional information on datasets and experiment setup

Datasets. We use five datasets for evaluation, with the dataset statistics shown in Table 2.

Table 2: Dataset statistics

Dataset	# Nodes	# Attributes	# Classes	# Test
Cora	2708	1433	7	1000
Pubmed	19717	500	3	1000
Friendster	43880	644	5	6251
Facebook	4556	3	2	1000
Protein	12679	29	2	2376

- **Cora** and **Pubmed** are benchmark datasets for node classification tasks from [30]. They are citation networks with nodes representing publications and edges representing citation relation. Node attributes are bag-of-word features of each document, and the predicted label is the corresponding research field.
- **Facebook** [36] is social network of Facebook users from Purdue university, where nodes represent users and edges represent friendship. The features of the nodes are: religious views, gender and whether the users hometown is in Indiana. The predicted labels is political view.
- **Friendster** [32] is social network. Nodes represent users and edges represent friendship. The node attributes include numerical features (e.g number of photos posted, etc) and categorical features (e.g. gender, college, music interests, etc), encoded as binary one-hot features. The node labels represent one of the five age groups: 0-24, 25-30, 36-40, 46-50 and over 50. This version of the graph contain 40K nodes, 25K of which are labeled.

- **Protein** is a collection of protein graphs from [1]. Each node is labeled with a functional role of the protein, and has a 29 dimensional feature vector. We use 85 graphs with an average size of 150 nodes.

Data splits. To conduct inductive learning tasks, we have to properly split the graphs into labeled and unlabeled parts. For datasets containing only one graph (Cora, Pubmed, Facebook and Friendster), we randomly sample a connected component to be $V_L^{(\text{tr})}$, and then sample a test set (V_T) from the remainder nodes (V_U). To make partially-labeled test data available, we sample another connected component as $V_L^{(\text{te})}$ with the same size as $V_L^{(\text{tr})}$. The nodes are sampled to guarantee that there is no overlapping between any two sets of $V_L^{(\text{tr})}$, $V_L^{(\text{te})}$ and V_T. Here $G^{(\text{tr})}$ and $G^{(\text{te})}$ have the same graph structure but with different labeled nodes.

For the protein dataset, as we have 85 disjoint graphs, we randomly choose 51 (60%) graphs for training, 17 (20%) graphs for validation and the remaining 17 (20%) graphs for testing. To simulate semi-supervised learning settings, we mask out 50% of true labels on the training graphs. For the tasks with partially-labeled test data, we randomly select 50% of the nodes in the test graphs as labeled nodes, and test on the remaining 50% nodes. We run five trials for all the experiments, and in each trial we randomly split the nodes/graphs as described.

As seen in Section 5.1, to approximate an inductive learning setting, we use a different train/test data split procedure (i.e. connected training set) on Cora and Pubmed networks from the public version (i.e. random training set) used in most of the existing GNN models [11, 15]. This is illustrated in Figure 3, where the random training set of the traditional GNN evaluation methods (in e.g., [11, 15]) is shown on the left, contrasted with our harder task of connected training set shown on the right. This difference in task is the reason why the model performance reported in our paper is not directly comparable with the reported results in previous GNN papers, even though we used the same implementations and hyperparameter search procedures.

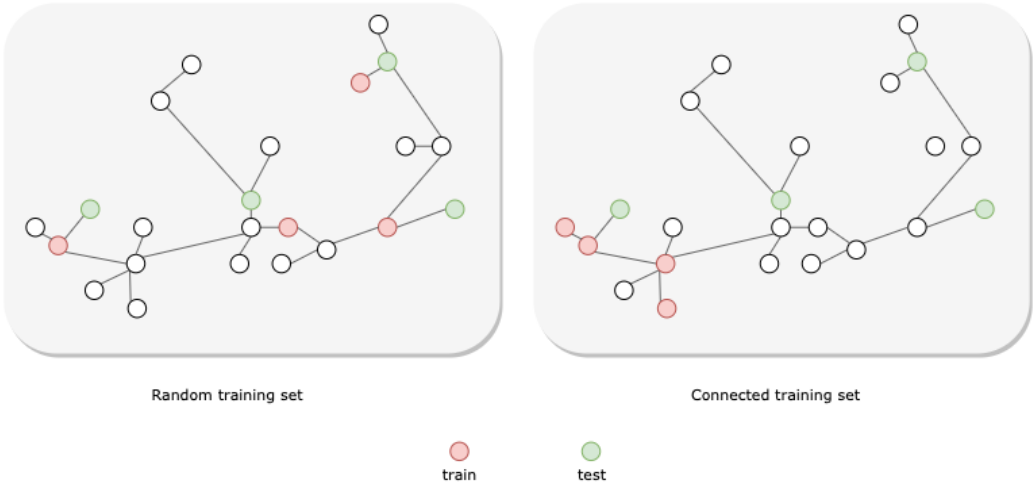


Figure 3: The different data splits between traditional GNN train/test split evaluation (left) and our—more realistic— connected train/random test split evaluation (right)

Hyperparameter setting. For hyperparameter tuning, we searched the initial learning rate within $\{0.005, 0.01, 0.05\}$ with weight decay of 0.0005. Dropout is applied to all the layers with $p = 0.5$. Hidden units are searched within $\{16, 32\}$ if the dataset wasn't used by the original GNN paper, or set as the same number as originally chosen in the GNN paper. The number of layers is set to 2 for both GCN [11] and GraphSage [7] as used in their paper, and we use 10 layers for TK [35]. For GraphSage [7], the neighborhood sample size is set to 5. We use the same GNN structure (i.e. layers, hidden units, neighborhood sample size) for the non-collective version and in *CL-GNN* for fair comparison.

For *CL-GNN*, the additional hyperparameters are (1) the sample size of predicted labels $\hat{Y}$ (K), and (2) the number of model iterations (T). we set sample size $K = 8$ for friendster dataset and

$K = 10$ for all other datasets. For label rate of 50%, the model is trained for $T = 10$ iterations, and each iteration contains 100 epochs. Note that we sample a new binary mask for each epoch as described in Section 3. For label rate of 0%, the model is trained for $T = 3$ iterations, and each iteration contains up to 500 epochs which can be early stopped if the validation accuracy decreases for a specified consecutive epochs. The numbers of iterations are empirically determined as only marginal improvements are observed after 3 iterations for unlabeled test data and 10 iterations for partially-labeled test data. The validation accuracy is used to choose the best epoch.

Note that the hyper-parameter tuning could be done more aggressively to further boost the performance of *CL-GNN*, e.g. using more layers for TK [35], but our main goal is to evaluate the relative improvements of *CL-GNN* on the corresponding non-collective GNNs.

F Running times for GNN models on multiple datasets

Dataset	GNN structure	Running time (minutes)		
		GNN	*CL-GNN*	
			unlabeled $G^{(\text{te})}$	partially-labeled $G^{(\text{te})}$
Cora	GCN	0.09	0.83	3.65
	TK	0.12	1.91	5.74
Pubmed	GCN	0.49	5.38	21.87
	TK	0.52	7.82	51.62
Friendster	GCN	1.04	17.93	66.31
	TK	1.93	30.17	132.33
Facebook	GCN	0.02	1.44	5.37
	TK	0.05	2.41	7.22

Table 3: The running time (in minutes) for *CL-GNN* and its corresponding GNNs.

Table 3 shows the running times for *CL-GNN* and the corresponding non-collective GNNs on various datasets. As mentioned in , for partially-labeled $G^{(\text{te})}$, *CL-GNN* applied random masks at each epoch, and ran for 10 iterations, whereas for unlabeled $G^{(\text{te})}$, *CL-GNN* ran for 3 iterations.

G *CL-GNN* performance with varying training label rates

To investigate the impact of the training label rates on the node classification accuracy, we repeated the experiments on Cora and Pubmed datasets with various numbers of training labels, on unlabeled test data and partially-labeled test data. Table 4 and Table 5 show the results for test labels rates of 0% and 50% respectively. We can see that in general *CL-GNN* framework achieved a larger improvement when fewer labels are available in the training graph. For example, with label rates of 1.52%, 1.90% and 3.04% on Pubmed, the improvements of our framework combining with GCN are 4.48%, 3.30% and 0.98% respectively. This shows that the *CL-GNN* framework is especially useful when only a small number of labels are available in training, which is the common use case of GNNs.

H Ablation study

H.1 With or without predicted labels as input

To investigate if adding predicted labels in model input adds extra information with partially-labeled test data, we tested the performance of a model variant which only use true labels as input with the same node masking procedure. Figure 4 shows two examples on Cora with GCN (Figure 4a) and Pubmed with TK (Figure 4b), where including predicted labels achieves better performance. We run the model 10 times and calculate the average and standard deviation (shown as shaded area) of classification accuracy at each iteration t as described in Section 3. We can see that adding predicted labels starts to improve the performance after the first iteration and achieves consistent gains.

		Cora			Pubmed		
# train labels		85 (3.21%)	105 (3.88%)	140 (5.17%)	300 (1.52%)	375 (1.90%)	600 (3.04%)
% test labels		0%	0%	0%	0%	0%	0%
Random		14.28 (0.00)	14.28 (0.00)	14.28 (0.00)	33.33 (0.00)	33.33 (0.00)	33.33 (0.00)
GCN	-	45.90 (3,26)	47.54 (3.50)	61.92 (1.50)	52.68 (2.36)	55.76 (3.32)	70.38 (2.31)
	+ CL	**+6.29 (1.49)**	**+5.20 (1.12)**	**+5.18 (0.66)**	+4.48 (2.33)	**+3.30(1.52)**	**+0.98(0.23)**
GS	-	50.69 (1.50)	56.24 (2.08)	66.08 (0.96)	59.34 (3.47)	64.37 (3.70)	72.08 (1.87)
	+ CL	**+2.35 (0.56)**	**+2.78 (0.59)**	**+1.95 (0.45)**	**+1.48 (0.41)**	**+0.62 (0.21)***	**+0.65 (0.25)**
TK	-	63.74 (2.61)	70.01 (1.93)	74.45 (0.34)	61.13 (5.03)	63.09 (5.57)	75.46 (1.46)
	+ CL	**+0.96 (0.30)***	**+1.08 (0.37)***	**+0.30 (0.11)***	**+1.00 (0.21)***	**+1.34 (0.20)**	**+1.03 (0.22)***
PL-EM[25]	-	20.70 (0.05)	24.65 (0.38)	30.46 (1.48)	38.05 (4.85)	44.85 (5.75)	51.25 (3.06)
ICA[14]	-	26.20 (0.51)	41.05 (0.50)	49.51 (1.90)	44.40 (1.92)	45.62 (0.86)	54.26 (2.09)
GMNN[28]	-	49.05 (1.86)	54.55 (1.15)	67.16 (1.86)	58.03 (3.62)	62.50 (3.77)	71.03 (4.54)

Table 4: Node classification accuracy with **unlabeled** test data varying number of **training labels** on Cora and Pubmed datasets. Numbers in bold represent significant improvement in a paired t-test at the $p < 0.05$ level, and numbers with * represent the best performing method in each column.

		Cora			Pubmed		
# train labels		85 (3.21%)	105 (3.88%)	140 (5.17%)	300 (1.52%)	375 (1.90%)	600 (3.04%)
% test labels		50%	50%	50%	50%	50%	50%
Random		14.28 (0.00)	14.28 (0.00)	14.28 (0.00)	33.33 (0.00)	33.33 (0.00)	33.33 (0.00)
GCN	-	36.38 (1.35)	48.31 (2.58)	64.02 (1.54)	54.11 (4.86)	56.31 (3.10)	68.13 (1.84)
	+ CL	**+15.69 (3.20)**	**+14.02 (3.38)**	**+6.31 (0.89)**	**+5.62 (1.17)**	**+5.06 (3.24)**	**+ 4.60 (2.50)**
GS	-	48.42 (2.82)	57.52 (2.15)	65.04 (0.79)	58.52 (5.42)	59.77 (4.68)	75.01 (4.86)
	+ CL	**+4.52 (0.84)**	**+3.06 (0.20)**	**+2.18 (0.21)**	**+2.42 (0.27)**	**+1.49 (0.10)**	**+2.67 (0.56)**
TK	-	55.68 (2.08)	61.51 (2.45)	67.95 (0.45)	63.05 (5.15)	67.95 (0.45)	74.01 (3.58)
	+ CL	**+7.18 (1.88)***	**+3.04 (1.07)***	**+2.75 (0.47)***	**+1.91 (0.75)***	+0.54 (0.44)*	**+3.23 (0.78)***
PL-EM[25]	-	20.35 (0.05)	25.25 (0.35)	31.45 (1.95)	31.70 (4.78)	34.92 (5.87)	48.70 (5.72)
ICA[14]	-	31.17 (3.66)	42.07 (1.29)	57.14 (1.81)	33.38 (4.69)	45.93 (5.48)	46.97 (5.19)
GMNN[28]	-	49.36 (2.22)	56.58 (2.96)	67.83 (1.91)	62.16 (4.40)	63.42 4.82)	74.78 (3.63)

Table 5: Node classification accuracy with **partially-labeled** test data varying number of **training labels** on Cora and Pubmed datasets. Numbers in bold represent significant improvement in a paired t-test at the $p < 0.05$ level, and numbers with * represent the best performing method in each column.

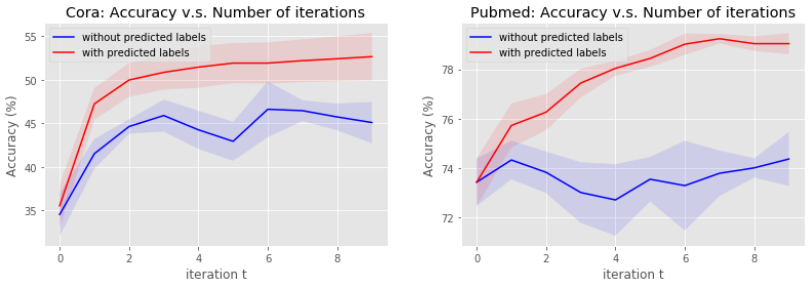
		Cora			Pubmed			Friendster	Facebook	Protein
# labels		85 (3.21%)	105 (3.88%)	140 (5.17%)	300 (1.52%)	375 (1.90%)	600 (3.04%)	641 (1.47%)	80 (1.76%)	7607 (30%)
Random		14.28 (0.00)	14.28 (0.00)	14.28 (0.00)	33.33 (0.00)	33.33 (0.00)	33.33 (0.00)	20.00 (0.00)	50.00 (0.00)	50.00 (0.00)
GCN	-	45.15 (3.73)	52.35 (2.01)	65.11 (1.95)	53.21 (4.04)	57.15 (3.61)	70.81 (3.47)	29.80 (0.48)	65.89 (0.68)	73.03 (2.14)
	+ CL-random	+0.02 (0.65)	-1.83 (1.05)	+0.27 (0.18)	**+2.29 (0.34)**	+1.35 (0.58)	+1.05 (0.96)	-0.14 (0.40)	**+1.16 (0.25)**	+0.16 (0.80)
GS	-	46.38 (1.62)	52.87 (1.03)	63.46 (1.38)	55.38 (3.48)	57.61 (4.21)	68.81 (4.15)	28.05 (0.56)	65.20 (0.40)	71.05 (0.40)
	+ CL-random	-2.45 (0.22)	+0.46 (0.45)	-0.23 (0.67)	-0.02 (0.32)	+0.42 (0.31)	+0.34 (0.53)	+0.21 (0.39)	**+1.65 (0.15)**	+0.01 (0.22)
TK	-	61.99 (3.07)	67.88 (1.80)	73.04 (0.42)	61.00 (4.93)	61.91 (5.16)	73.87 (3.99)	29.44 (0.39)	67.75 (0.40)	73.38 (0.57)
	+ CL-random	-3.95 (1.08)	-2.54 (0.63)	-2.28 (0.84)	-0.65 (0.56)	-0.78 (0.38)	-1.17 (0.78)	+0.26 (0.38)	-0.05 (0.19)	-0.13 (0.53)

Table 6: Node classification accuracy with **unlabeled** test data using **uniform sampling**. Numbers in bold represent significant improvement in a paired t-test at the $p < 0.05$ level.

H.2 Sampling from predicted labels or random ids

Creating more expressive GNN representations by averaging out random features was first proposed by Murphy et al. [20]. Murphy et al. [20] shows a whole-graph classification application, Circulant Skip Links (CSL) graphs, where such randomized feature averaging is provably (and empirically) more expressive than GNNs. Our Monte Carlo collective learning method can be seen as a type of feature averaging GNN representation though, unlike Murphy et al. [20], the feature sampling is not at random, but rather driven by our own model recursively. Hence, it is fair to ask if our performance gains are simply because random feature averaging is beneficial to GNN representations? Or does collective learning sampling actually improve performance? We need an ablation study.

Therefore, in this section we investigate whether the gains of our method for unlabeled test data are from incorporating feature randomness, or from sampling w.r.t predicted labels (collective learning). To do so, we replace the samples drawn from previous prediction $\hat{\boldsymbol{Y}}$ as uniformly drawn from the set of class labels *at each gradient step* in *CL-GNN*. The results are shown in Table 6. Clearly, the



(a) Cora, GCN (b) Pubmed, TK

Figure 4: *CL-GNN* performance with and without predicted labels on Cora and Pubmed. X-axis refers to iteration number t in Section 3

random features are not able to consistently improve the model performance as our method does (contrast Table 6 with Table 1 and Table 4). In summary, collective learning goes beyond the purely randomized approach of Murphy et al. [20], providing much larger, statistically significant, gains.