Graph Few-shot Learning via Knowledge Transfer

Huaxiu Yao[1], **Chuxu Zhang**[2], **Ying Wei**[3], **Meng Jiang**[2],
Suhang Wang[1], **Junzhou Huang**[3], **Nitesh V. Chawla**[2], **Zhenhui Li**[1]
[1]Pennsylvania State University, [2]University of Notre Dame, [3]Tencent AI Lab
[1]{huaxiuyao,szw494,zul17}@psu.edu,[2]{czhang11,mjiang2,abotello}@nd.edu
[3]judyweiying@gmail.com, joehhuang@tencent.com

Abstract

Towards the challenging problem of semi-supervised node classification, there have been extensive studies. As a frontier, Graph Neural Networks (GNNs) have aroused great interest recently, which update the representation of each node by aggregating information of its neighbors. However, most GNNs have shallow layers with a limited receptive field and may not achieve satisfactory performance especially when the number of labeled nodes is quite small. To address this challenge, we innovatively propose a graph few-shot learning (GFL) algorithm that incorporates prior knowledge learned from auxiliary graphs to improve classification accuracy on the target graph. Specifically, a transferable metric space characterized by a node embedding and a graph-specific prototype embedding function is shared between auxiliary graphs and the target, facilitating the transfer of structural knowledge. Extensive experiments and ablation studies on four real-world graph datasets demonstrate the effectiveness of our proposed model.

1 Introduction

Classifying a node (e.g., predicting interests of a user) in a graph in a semi-supervised manner has been challenging but imperative, inasmuch as only a small fraction of nodes have access to annotations which are usually costly. Recently, graph neural networks (GNN) [7, 14] have attracted considerable interest and demonstrated promising performance. To their essential characteristics, GNNs recursively update the feature of each node through aggregation (or message passing) of its neighbors, by which the patterns of graph topology and node features are both captured. Nevertheless, considering that adding more layers increases the difficulty of training and over-smoothens node features [7], most of existing GNNs have shallow layers with a restricted receptive field. Therefore, GNNs are inadequate to characterize the global information, and work not that satisfactorily when the number of labeled nodes is especially small.

Inspired by recent success of few-shot learning, from a innovative perspective, we are motivated to *leverage the knowledge learned from auxiliary graphs to improve semi-supervised node classification in the target graph of our interest.* The intuition behind lies in that auxiliary graphs and the target graph likely share local topological structures as well as class-dependent node features [11, 8]. Yet it is even more challenging to achieve few-shot learning on graphs than on i.i.d. data (e.g., images) which exisiting few shot learning algorithms focus on. The two lines of recent few-shot learning works, including gradient-based methods [4, 10] and metric-based methods [12, 15], formulate the transferred knowledge as parameter initializations (or a meta-optimizer) and a metric space, respectively. None of them, however, meets the crucial prerequisite of graph few-shot learning to succeed, i.e., transferring underlying structures across graphs.

To this end, we propose a novel **G**raph **F**ew-shot **L**earning (GFL) model. Built upon metric-based few-shot learning, the basic idea of GFL is to learn a transferable metric space in which the label of a node is predicted as the class of the nearest prototype to the node. The metric space is practically characterized with two embedding functions, which embed a node and the prototype of each class, respectively. Specifically, first, GFL learns the representation of each node using a graph autoencoder

NeurIPS 2019 Graph Representation Learning Workshop (GRL 2019), Vancouver, Canada.

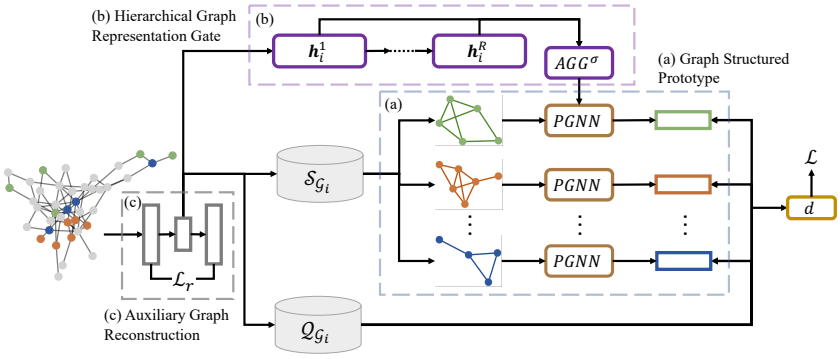


Figure 1: The framework of proposed GFL.

whose backbone is GNNs. Second, to better capture global information, we establish a relational structure of all examples belonging to the same class, and learn the prototype of this class by applying a prototype GNN to the relational structure. Most importantly, both embedding functions encrypting structured knowledge are transferred from auxiliary graphs to the target one, to remedy the lack of labeled nodes. Besides the two node-level structures, note that we also craft the graph-level representation via a hierarchical graph representation gate, to enforce that similar graphs have similar metric spaces. The experiments on node classification problem demonstrate the effectiveness of GFL.

2 Preliminaries

Graph Neural Network A graph $\mathcal{G}$ is represented as $(\mathbf{A}, \mathbf{X})$, where $\mathbf{A} \in \{0,1\}^{n\times n}$ is the adjacent matrix, and $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n\times h}$ is the node feature matrix. To learn the node represetation for graph $\mathcal{G}$, an embedding function f with parameter θ are defined as:

$$\mathbf{H}^{(l+1)} = \mathcal{M}(\mathbf{A}, \mathbf{H}^{(l)}; \mathbf{W}^{(l)}), \tag{1}$$

where $\mathcal{M}$ is the message passing function. After stacking L graph neural network layers, we can get the final representation $\mathbf{Z} = \mathrm{GNN}(\mathbf{A}, \mathbf{X}) = f_\theta(\mathbf{A}, \mathbf{X}) = \mathbf{H}^{(L+1)} \in \mathbb{R}^{h'}$.

The Graph Few-Shot Learning Problem In graph few-shot learning, we are given a sequence of graphs $\{\mathcal{G}_1, \ldots, \mathcal{G}_{N_t}\}$ sampled from a probability distribution $\mathcal{E}$ over tasks [3]. For each graph $\mathcal{G}_i \sim \mathcal{E}$. we are provided with a small set of n^{s_i} labeled *support* nodes set $\mathcal{S}_i = \{(\mathbf{x}_{i,j}^{s_i}, y_{i,j}^{s_i})\}_{j=1}^{n^{s_i}}$ and a *query* nodes set $\mathcal{Q}_i = \{(\mathbf{x}_{i,j}^{q_i}, y_{i,j}^{q_i})\}_{j=1}^{n^{q_i}}$. For each node j in query set $\mathcal{Q}_i$, we are supposed to predict its corresponding label by associating its embedding $f_\theta(\mathbf{A}, \mathbf{x}_{i,j}^{q_i}) : \mathbb{R}^h \to \mathbb{R}^{h'}$ with representation $(f_\theta(\mathbf{A}, \mathbf{x}_{i,j}^{s_i}), y_{i,j}^{s_i})$ in support set $\mathcal{S}_i$ via the similarity measure d. Specifically, in prototypical network [12], the prototype $\mathbf{c}_i^k$ for each class k is defined as $\mathbf{c}_i^k = \sum_{\mathbf{x}_{i,j}^{s_i} \in \mathcal{S}_i^k} f_\theta(\mathbf{A}, \mathbf{x}_{i,j}^{s_i})/|\mathcal{S}_i^k|$, where $\mathcal{S}_i^k$ denotes the sample set in $\mathcal{S}_i$ of class k and $|\mathcal{S}_i^k|$ means the number of samples in $\mathcal{S}_i^k$. For each graph $\mathcal{G}_i$, the effectiveness on query set $\mathcal{Q}_i$ is evaluated by the loss $\mathcal{L}_i = \sum_k \mathcal{L}_i^k$, where:

$$\mathcal{L}_i^k = -\sum_{(\mathbf{x}_{i,j}^{q_i}, y_{i,j}^{q_i}) \in \mathcal{Q}_i^k} \log \frac{\exp(-d(f_\theta(\mathbf{A}, \mathbf{x}_{i,j}^{q_i}), \mathbf{c}_i^k))}{\sum_{k'} \exp(-d(f_\theta(\mathbf{A}, \mathbf{x}_{i,j}^{q_i}), \mathbf{c}_i^{k'}))}, \tag{2}$$

where $\mathcal{Q}_i^k$ is the query set of class k from $\mathcal{Q}_i$. To achieve this goal, few-shot learning often includes two-steps, i.e., meta-training and meta-testing. In meta-training, the parameter θ of embedding function f_θ is optimized to minimize the expected empirical loss over all historical training graphs, i.e., $\min_\theta \sum_{i=1}^{N_t} \mathcal{L}_i$. Once trained, given a new graph $\mathcal{G}_t$, the learned embedding function f_θ can be used to improve the learning effectiveness with a few support nodes.

3 Methodology

In this section, we elaborate our proposed GFL whose framework is illustrated in Figure 1. We will detail the three components of GFL, i.e., *graph structured prototype*, *hierarchical graph representation gate* and *auxiliary graph reconstruction*.

Graph Structured Prototype In most of the cases, a node plays two important roles in a graph: one is locally interacting with the neighbors that may belong to different classes; the other is interacting with the nodes of the same class in relatively long distance, which can be globally observed. It

is non-trivial to model the relational structure among support nodes and learn their corresponding prototype, we thus propose a prototype GNN model denoted as PGNN to tackle this challenge.

Given the representation of each node, we first extract the relational structure of samples belong to class k. For each graph $\mathcal{G}_i$, the relational structure $\mathcal{R}_i^k$ of the sample set $\mathcal{S}_i^k$ can be constructed based on the number of k-hop common neighbors. Then, the PGNN is used to model the interactions between samples in the $\mathcal{S}_i^k$, i.e., $\mathrm{PGNN}_\phi(\mathcal{R}_i^k, f_\theta(\mathcal{S}_i^k))$, where PGNN is parameterized by ϕ and then we use j to indicate the j-th node representation (see part (a) in Figure 1).

$$\mathbf{c}_i^k = \mathrm{Pool}_{j=1}^{n^{s_i^k}}(\mathrm{PGNN}_\phi(\mathcal{R}_i^k, f_\theta(\mathcal{S}_i^k))[j]), \tag{3}$$

where Pool operator denotes a max or mean pooling operator over support nodes and $n^{s_i^k}$ represents the number of nodes in support set $\mathcal{S}_i^k$.

Hierarchical Graph Representation Gate The above prototype construction process is highly determined by the PGNN with the globally shared parameter ϕ. However, different graphs have their own topological structures, motivating us to tailor the globally shared information to each graph. Thus, we learn a hierarchical graph representation for extracting graph-specific information and incorporate it with the parameter of PGNN through a gate function (see part (b) in Figure 1 and more detailed structure is in Appendix A). Following [18], the hierarchical graph representation for each level is accomplished by alternating between two level-wise stages:

I. Node Assignment In the assignment step, each low-level node is assigned to high-level community. In level r , we denote the number of nodes as K^r, the adjacency matrix as $\mathbf{A}_i^r$, the feature matrix as $\mathbf{X}_i^r$. The assignment matrix $\mathbf{P}_i^{r\to r+1} \in \mathbb{R}^{K^r \times K^{r+1}}$ from level r to level $r+1$ is calculated by applying softmax function on the output of an assignment GNN (AGNN) as follows:

$$\mathbf{P}_i^{r\to r+1} = \mathrm{Softmax}(\mathrm{AGNN}(\mathbf{A}_i^r, \mathbf{X}_i^r)), \tag{4}$$

II. Representation Fusion After getting the assignment matrix , for level $r+1$, the adjacent matrix is defined as $\mathbf{A}_i^{r+1} = (\mathbf{P}_i^{r\to r+1})^T \mathbf{A}_i^r \mathbf{P}_i^{r\to r+1}$ and the feature matrix is calculated by applying assignment matrix on the output of a fusion GNN (FGNN), i.e., $\mathbf{X}_i^{r+1} = (\mathbf{P}_i^{r\to r+1})^T \mathrm{FGNN}(\mathbf{A}_i^r, \mathbf{X}_i^r)$. Then, the feature representation $\mathbf{h}_i^{r+1}$ of level $r+1$ can be calculated as:

$$\mathbf{h}_i^{r+1} = \mathrm{Pool}_{k^{r+1}=1}^{K^{r+1}}((\mathbf{P}_i^{r\to r+1})^T \mathrm{FGNN}(\mathbf{A}_i^r, \mathbf{X}_i^r)[k^{r+1}]), \tag{5}$$

Then, to get the whole graph-specific representation $\mathbf{h}_i$, the representation of each level is aggregated via an aggregator AGG, i.e., $\mathbf{h}_i = \mathrm{AGG}(\mathbf{h}_i^1, \ldots, \mathbf{h}_i^R)$. Both mean pooling aggregator and attention aggregator are used in this paper. Inspired by previous findings [16]: similar graphs may activate similar parameters (i.e., parameter ϕ of the PGNN), we introduce a gate function $\mathbf{g}_i = \mathcal{T}(\mathbf{h}_i)$ to tailor graph structure specific information. Then, the global transferable knowledge (i.e., ϕ) is adapted to the structure-specific parameter via the gate function, i.e., $\phi_i = \mathbf{g}_i \circ \phi = \mathcal{T}(\mathbf{h}_i) \circ \phi$ where $\circ$ represents element-wise multiplication. $\mathbf{g}_i = \mathcal{T}(\mathbf{h}_i) = \sigma(\mathbf{W}_g \mathbf{h}_i + \mathbf{b}_g)$ maps the graph-specific representation $\mathbf{h}_i$ to the same space of parameter ϕ. Thus, PGNN_ϕ in Eqn. (3) would be PGNN_{ϕ_i}.

Auxiliary Graph Reconstruction In practice, it is difficult to learn an informative node representation using only the signal from the matching loss, which motivates us to design a new constraint for improving the training stability and the quality of node representation (see part (c) in Figure 1). Thus, for the node embedding function, we refine it by using a graph autoencoder and the reconstruction loss is defined as follows,

$$\mathcal{L}_r(\mathbf{A}_i, \mathbf{X}_i) = \|\mathbf{A}_i - \mathrm{GNN}_{dec}(\mathbf{Z}_i)\mathrm{GNN}_{dec}^T(\mathbf{Z}_i)\|_F^2, \tag{6}$$

where $\mathbf{Z}_i = \mathrm{GNN}_{enc}(\mathbf{A}_i, \mathbf{H}_i)$ is the representation for each node. Recalling the objective in Section 2, we reach the optimization problem of GFL as $\min_\Theta \sum_{i=1}^{N_t} \mathcal{L}_i + \gamma \mathcal{L}_r(\mathbf{A}_i, \mathbf{X}_i)$, where Θ represents all learnable parameters. The whole training process (algorithm) of GFL is detailed in Appendix B.

4 Experiments

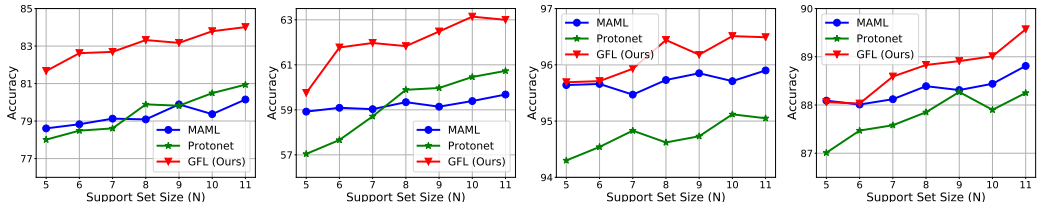
Dataset and Experimental Settings We evaluate our model performance on four tasks: (1) classifying 4 research domains of authors using AMiner collaboration data [2]; (2) classifying 5 communities of posts using Reddit posts data [6]; (3) classifying 3 categories of papers using AMiner citation data [2]; (4) classifying 3 subjects of papers using PubMed data [14]. The details of these datasets are summarized in Appendix C. In this work, we follow the traditional K-way N-shot few-shot learning settings [4, 12]. For each graph, N labeled nodes for each class are provided. The rest nodes are used as query set. Like [7], the embedding structure is a two-layer graph convolutional structure (GCN) with 32 neurons in each layer. The distance metric d is defined as the inner product distance (see Appendix D for detailed hyperparameter settings).

Table 1: Comparison between GFL and other node classification methods on four graph datasets. Performance of Accuracy$\pm 95\%$ confidence intervals on 10-shot classification are reported.

Model	Collaboration	Reddit	Citation	Pubmed
LP [19]	$61.09 \pm 1.36\%$	$23.40 \pm 1.63\%$	$67.00 \pm 4.50\%$	$48.55 \pm 6.01\%$
Planetoid [17]	$62.95 \pm 1.23\%$	$50.97 \pm 3.81\%$	$61.94 \pm 2.14\%$	$51.43 \pm 3.98\%$
Deepwalk [9]	$51.74 \pm 1.59\%$	$34.81 \pm 2.81\%$	$56.56 \pm 5.25\%$	$44.33 \pm 4.88\%$
node2vec [5]	$59.77 \pm 1.67\%$	$43.57 \pm 2.23\%$	$54.66 \pm 5.16\%$	$41.89 \pm 4.83\%$
Base-GCN [7]	$63.16 \pm 1.47\%$	$46.21 \pm 1.43\%$	$63.95 \pm 5.93\%$	$54.87 \pm 3.60\%$
Finetune	$76.09 \pm 0.56\%$	$54.13 \pm 0.57\%$	$88.93 \pm 0.72\%$	$83.06 \pm 0.72\%$
K-NN	$67.53 \pm 1.33\%$	$56.06 \pm 1.36\%$	$78.18 \pm 1.70\%$	$74.33 \pm 0.52\%$
Matchingnet [15]	$80.87 \pm 0.76\%$	$56.21 \pm 1.87\%$	$94.38 \pm 0.45\%$	$85.65 \pm 0.21\%$
MAML [4]	$79.37 \pm 0.41\%$	$59.39 \pm 0.28\%$	$95.71 \pm 0.23\%$	$88.44 \pm 0.46\%$
Protonet [12]	$80.49 \pm 0.55\%$	$60.46 \pm 0.67\%$	$95.12 \pm 0.17\%$	$87.90 \pm 0.54\%$
GFL-mean (Ours)	$83.51 \pm 0.38\%$	$62.66 \pm 0.57\%$	$\mathbf{96.51 \pm 0.31}\%$	$\mathbf{89.37 \pm 0.41}\%$
GFL-att (Ours)	$\mathbf{83.79 \pm 0.39}\%$	$\mathbf{63.14 \pm 0.51}\%$	$95.85 \pm 0.26\%$	$88.96 \pm 0.43\%$

Baseline Methods For performance comparison of node classification, we consider three types of baselines: (1) *Graph-based semi-supervised methods* including Label Propagation (LP) [19] and Planetoid [17]; (2) *Graph representation learning methods* including Deepwalk [9], node2vec [5] and Base-GCN [7]. Note that, for Base-GCN, we train GCN on each meta-testing graph with limited labeled data rather than transferring knowledge from meta-training graphs; (3) *Transfer/few-shot methods* including finetune, K-nearest-neighbor (K-NN), Matching Network (Matchingnet) [15], MAML [4], Prototypical Network (Protonet) [12]. Each transfer/few-shot learning method uses the same embedding structure as GFL. Detailed description of baselines can be found in Appendix E.

Overall Results For each dataset, we reported the averaged accuracy with 95% confidence interval over meta-testing graphs of 10-shot node classification in Table 1. Both GFL-mean and GFL-att achieve the best performance than all three types of baselines on four datasets, indicating the effectiveness by incorporating graph prototype and hierarchical graph representation. In addition, as a metric distance based meta-learning algorithm, GFL not only outperforms other algorithms from this research line (i.e., Matchingnet, Protonet), but also achieves better performance than MAML, a popular gradient-based meta-learning algorithm. We also conduct extensive ablation studies and sensitivity analysis, which are reported in Appendix F and G, respectively.



(a) : Collaboration (b) : Reddit (c) : Citation (d) : Pubmed

Figure 2: Effect of support set size, which is represented by the shot number N

Effect of Support Set Size In addition, we analyze the effect of support set size, which is represented by the shot number N. We select two representative few-shot learning methods: Protonet (metric-learning based model) and MAML (gradient-based model). The results of each dataset are shown in Figure 2a-2d. We can see when the support set size is small, Protonet performs worse than MAML. The potential reason is that calculating prototype by averaging values over samples may be sensitive to outliers. Thus, it requires more data to reduce the effect of outliers. By extracting the relational structure among samples of the same class, our algorithm is more robust and achieves best performance in all scenarios.

5 Conclusion

In this paper, we introduce a new framework GFL to improve the learning effectiveness on a new graph by transferring knowledge from previous learned graphs. GFL improves metric-based few-shot learning by integrating graph structure from local node-level to global graph-level. We conduct extensive experiments and the results demonstrate the effectiveness of our proposed model on four node classification tasks.

References

[1] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[2] Aminer. `https://aminer.org/`, 2019.

[3] Jonathan Baxter. Theoretical models of learning to learn. In *Learning to learn*, pages 71–94. Springer, 1998.

[4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, 2017.

[5] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864. ACM, 2016.

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[7] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[8] Danai Koutra, Vogelstein Joshua T., and Christos Faloutsos. Deltacon: A principled massive-graph similarity function. In *SDM*, 2013.

[9] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710, 2014.

[10] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *ICLR*, 2016.

[11] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 12(Sep):2539–2561, 2011.

[12] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NIPS*, pages 4077–4087, 2017.

[13] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[14] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[15] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NIPS*, pages 3630–3638, 2016.

[16] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, pages 2048–2057, 2015.

[17] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, pages 40–48, 2016.

[18] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NIPS*, pages 4805–4815, 2018.

[19] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Citeseer, 2002.

A Detailed Framework of Hierarchical Graph Representation Gate

In Figure 3, we illustrate the detailed framework of hierarchical graph representation gate (detailed description of this component is in Section 4.2). Part (a) shows the basic block for learning hierarchical representation $\{\mathbf{h}_i^1, \ldots, \mathbf{h}_i^R\}$. The aggregator is illustrated in part (b), where the learnable query vector $\mathbf{q}_i$ is introduced to calculate the attention weight $\beta^1 \cdots \beta^R$. Note that, we only illustate the attention aggregator. For mean pooling aggregator, we calculate the average value of $\mathbf{h}_i^1, \cdots, \mathbf{h}_i^R$ as the graph representation $\mathbf{h}_i$. Then, the graph representation $\mathbf{h}_i$ is used to calculate gate $\mathbf{g}_i$ by using a fully connected layer with sigmoid activation in part (c).

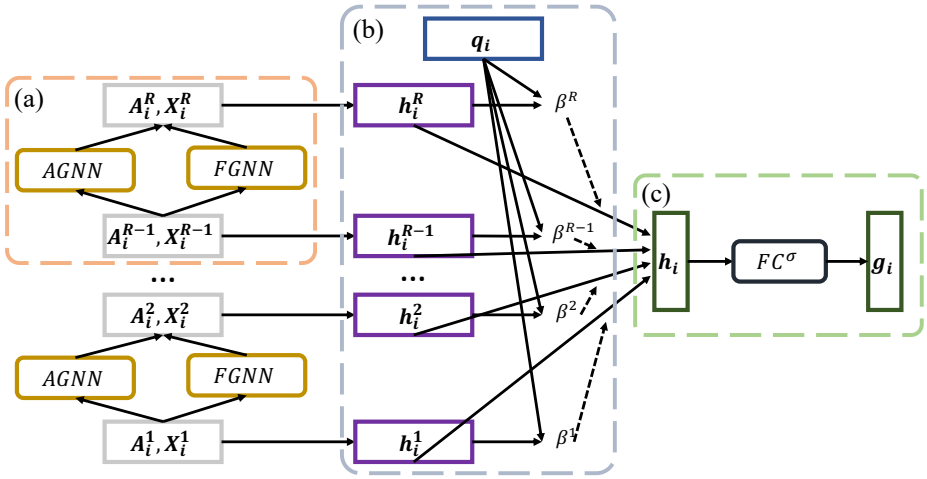


Figure 3: The detailed framework of hierarchical graph representation gate: (a) basic block for representation learning; (b) aggregator to aggregate hierarchical representations; (c) graph representation gate construction.

B Meta-training Process of GFL

Algorithm 1 Training Process of GFL

Require: $\mathcal{E}$: distribution over graphs; L: # of layers in hierarchical structure; α: stepsize; γ: balancing parameter for loss

Randomly initialize Θ
while not done **do**
  Sample a batch of graphs $\mathcal{G}_i \sim \mathcal{E}$ and its corresponding adjacent matrices $\mathbf{A}_i$ and feature matrices $\mathbf{X}_i$
  for all $\mathcal{G}_i$ **do**
    Sample support set $\mathcal{S}_i$ and query set $\mathcal{Q}_i$
    Compute the embedding $f_\theta(\mathbf{A}_i, \mathbf{X}_i)$ and its reconstruction error $\mathcal{L}_r(\mathbf{A}_i, \mathbf{X}_i)$ in Eqn. (6)
    Compute the hierarchical representation $\{\mathbf{h}_i^1, \ldots, \mathbf{h}_i^R\}$ in Eqn. (5) and gate $\mathbf{g}_i$
    Construct relational graphs $\{\mathcal{R}_i^1, ..., \mathcal{R}_i^K\}$ for samples in $\mathcal{S}_i$ and compute graph prototype $\{\mathbf{c}_i^1, ..., \mathbf{c}_i^K\}$ in Eqn. (3).
    Compute the matching score using the query set $\mathcal{Q}_i$ and evaluate loss in Eqn. (2)
  end for
  Update $\Theta \leftarrow \Theta - \alpha \nabla_\Theta \sum_{i=1}^{N_t} \mathcal{L}_i(\mathbf{A}_i, \mathbf{X}_i) + \gamma \mathcal{L}_r(\mathbf{A}_i, \mathbf{X}_i)$
end while

C Detailed Dataset Description

We use four datasets of different kinds of graphs: Collaboration, Reddit, Citation and Pubmed. (1): *Collaboration data*: Our first task is to predict research domains of different academic authors. We

use the collaboration graphs extracted from the AMiner data [2]. Each author is assigned with a computer science category label according to the majority of their papers' categories. (2): *Reddit data*: In the second task, we predict communities of different Reddit posts. We construct post-to-post graphs from Reddit community data [6], where each edge denotes that the same user comments on both posts. Each post is labeled with a community id. (3): *Citation data*: The third task is to predict paper categories. We derive paper citation graphs from the AMiner data and each paper is labeled with a computer science category label. (4): *Pubmed data*: Similar to the third task, the last task is to predict paper class labels. The difference is that the citation graphs are extracted from the PubMed database [14] and each node is associated with diabetes class id. The statistics of these datasets are reported in Table 2.

Table 2: Data Statistics.

Dataset	Collaboration	Reddit	Citation	Pubmed
# Nodes (avg.)	4,496	5,469	2,528	2,901
# Edges (avg.)	14,562	7,325	14,710	5,199
# Features/Node	128	600	100	500
# Classes	4	5	3	3
# Graphs (Meta-training)	100	150	30	60
# Graphs (Meta-validation)	10	15	3	5
# Graphs (Meta-testing)	20	25	10	15

D Detailed Hyperparameter Settings

For more detailed hyperparameter settings, the learning rate is set as 0.01, the dimension of hierarchical graph representation $\mathbf{h}_i$ is set as 32. For the hierarchical graph representation learning structure, we follow the strategy in [18] that the number of nodes in a higher layer is half of that in its consecutive lower layer. The specific values for the number of nodes in the 1st layer and the number of layers are determined by the tuning process on the validation set. In our experiments, the number of hierarchical layer R is set as 3 and the number of nodes in layer 2 and layer 3 are set as 64 and 32, respectively. The reconstruction loss weight γ is set as 1.0. Additionally, in order to construct the relational graph of few-shot labelled nodes for each class, we compute the similarity score between each two nodes by counting the number of k-hop (k=3) common neighbors and further smooth this similarity value by a sigmoid function. The computed similarity matrix is further fed into GNN for generating prototype embedding. We implement all experiments using Pytorch[1].

E Detailed Descriptions of Baselines

We detail three types of baselines in this section. Note that, all GCN used in baselines are two-layers GCN with 32 neurons each layer. The descriptions are as follows:

- **Graph-based semi-supervised methods**:
  - **Label Propagation (LP)** [19]: Label Propagation is a traditional semi-supervised learning methods.
  - **Planetoid** [17] Planetoid is a semi-supervised learning method based on graph embeddings. We use transductive formulation of Planetoid in this paper.
- **Graph representation learning methods**:
  - **Deepwalk** [9]: Deepwalk learns the node embedding in an unsupervised way. We concatenate the learned node embedding and the node features, then feed them to the multiclass classifier (using Scikit-Learn[2]). Few-shot node labels in each graph are available for training classifer.
  - **node2vec** [5]: This method is similar to the Deepwalk while we use node2vec model to learn node embedding.
  - **Non-transfer-GCN** [7] In Non-transfer-GCN, we only train GCN on each meta-testing network without transferring knowledge from meta-training networks.

[1] https://pytorch.org

[2] https://scikit-learn.org/stable/

- **Transfer/Few-shot methods**
  - **All-Graph-Finetune (AGF)** In AGF, we train GCN by feeding each meta-training graph one-by-one. Then, we can learn the initialization of GCN parameters from meta-training graphs. In meta-testing process, we finetune the learned initialization on every meta-testing graphs. The hyperparameters (e.g., learning rate) of AGF are the same as GFL.
  - **K-nearest-neighbor** (K-NN) Similar as the settings of [13], we first learn the initialization of GCN parameters by using all meta-training graphs. Then, in the testing process, we use the learned embedding function (i.e., GCN) to learn the representation of support nodes and each query node. Finally, we use k-NN for classification.
  - **Matching Network (Matchingnet)** [15]: Matching network is a metric-based few-shot learning method. Since traditional matching network focuses on one-shot learning, in our scenario, we still use each query node representation to match the most similar on in support set and use its label as the predicted label for the query node.
  - **MAML** [4]: MAML is a representative gradient-based meta-learning method, which learn a well-generalized model initialization which can be adapted with a few gradient steps. For MAML, we set the learning rate of inner loop as 0.001.
  - **Prototypical Network (Protonet)** [12] Prototypical network is a representative metric-based few-shot learning method, which constructs the prototype by aggregating nodes belong to the same class using mean pooling.

F Ablation Studies

Since GFL integrates three essential components (i.e., graph structured prototype, hierarchical graph representation gate, auxiliary graph reconstruction), we conduct extensive ablation studies to understand the contribution of each component. Table 3 shows the results of ablation studies on each dataset, where the best results among GFL-att and GFL-mean are reported as GFL results. Performance of accuracy are reported in this table. For the graph structured prototype, in (M1a), we first report the performance of protonet for comparison since Protonet use mean pooling of node embedding instead of constructing and exploiting relational structure for each class.

To show the effectiveness of hierarchical graph representation gate, we first remove this component and report the performance in (M2a). The results are inferior, demonstrating that the effectiveness of graph-level representation. In addition, we only use the flat representation structure (i.e., $R = 1$) in (M2b). The results show the effectiveness of hierarchical representation.

For auxiliary graph reconstruction, we remove the decoder GNN and only use the encoder GNN to learn the node representation in (M3). GFL outperforms (M3) as the graph reconstruction loss refines the learned node representation and enhance the stability of training.

Table 3: Results of Ablation Studies. Accuracy scores on 10-shot node classification are reported. We select the best performance of GFL-mean and GFL-att for GFL in this table.

Ablation Model	Collab.	Reddit	Citation	Pubmed
(M1a): use the mean pooling prototype (i.e., protonet)	80.49%	60.46%	95.12%	87.90%
(M1b): replace the gate with the Film modulation	82.86%	62.66%	95.42%	88.92%
(M2a): remove the hierarchical representation gate	82.63%	61.99%	95.33%	88.15%
(M2b): use flat representation rather than hierarchical way	83.45%	62.55%	95.76%	89.08%
(M3): remove the graph reconstruction loss	82.98%	62.58%	95.63%	89.11%
GFL (Ours)	**83.79%**	**63.14%**	**96.51%**	**89.37%**

G Sensitivity Analysis

G.1 Effect of Distance Functions d

In addition, we replace the distance function d in Eqn. (2) from inner product (used in the original setting) for cosine distance. The results are reported in Table 4. Compared with inner product, the similar results show that GFL is not very sensitivity to the distance function d.

Table 4: Effect of different distance functions d.

Method	Collab.	Reddit	Cita.	Pubmed
GFL (inner product)	83.79%	63.14%	96.51%	89.37%
GFL (cosine)	84.02%	62.95%	96.02%	89.25%

G.2 Effect of Different Similarity Functions for Constructing Relational Structure $\mathcal{R}_i^k$

We further analyze the effect of different similarity functions for constructing relational structure of the graph prototype. Jaccard Index, Adamic-Adar [1], PageRank and Top-k Common Neighbors (Top-k CN) are selected and the results are reported in Table 5. Note that, in previous results, we all use Top-k CN as similarity function. The results show that GFL is not very sensitive to the similarity on a dataset may be achieved by different functions.

Table 5: Effect of different similarity functions for calculating relational structure $\mathcal{R}_i^k$. Results of Accuracy are reported.

Method	Collab.	Reddit	Cita.	Pubmed
Jaccard	82.98%	62.71%	95.18%	88.91%
Adamic-Adar	83.70%	62.87%	95.49%	89.21%
PageRank	**84.14%**	63.08%	95.93%	**90.02%**
Top-k CN	83.79%	**63.14%**	**96.51%**	89.37%