Adaptive-Step Graph Meta-Learner for Few-Shot Graph Classification

Ning Ma[1,2,3], Jiajun Bu[1,2,3,*], Jieyu Yang[1,2,3], Zhen Zhang[1,2,3]
Chengwei Yao[1,2,3], Zhi Yu[1,2,3], Sheng Zhou[1,2,3], Xifeng Yan[4]
[1]Zhejiang Provincial Key Laboratory of Service Robot, College of Computer Science, Zhejiang University
[2] Alibaba-Zhejiang University Joint Institute of Frontier Technologies
[3] Ningbo Research Institute, Zhejiang University
[4] Computer Science Department, University of California Santa Barbara
{ma_ning,bjj,yangjieyu,zhen_zhang,yaochw,yuzhirenzhe,zhousheng_zju}@zju.edu.cn,xyan@cs.ucsb.edu

ABSTRACT

Graph classification aims to extract accurate information from graph-structured data for classification and is becoming more and more important in the graph learning community. Although Graph Neural Networks (GNNs) have been successfully applied to graph classification tasks, most of them overlook the scarcity of labeled graph data in many applications. For example, in bioinformatics, obtaining protein graph labels usually needs laborious experiments. Recently, few-shot learning has been explored to alleviate this problem with only a few labeled graph samples of test classes. The shared sub-structures between training classes and test classes are essential in the few-shot graph classification. Existing methods assume that the test classes belong to the same set of super-classes clustered from training classes. However, according to our observations, the label spaces of training classes and test classes usually do not overlap in a real-world scenario. As a result, the existing methods don't well capture the local structures of unseen test classes. To overcome the limitation, in this paper, we propose a direct method to capture the sub-structures with a well initialized meta-learner within a few adaptation steps. More specifically, (1) we propose a novel framework consisting of a graph meta-learner, which uses GNNs based modules for fast adaptation on graph data, and a step controller for the robustness and generalization of meta-learner; (2) we provide quantitative analysis for the framework and give a graph-dependent upper bound of the generalization error based on our framework; (3) the extensive experiments on real-world datasets demonstrate that our framework gets state-of-the-art results on several few-shot graph classification tasks compared to baselines.

CCS CONCEPTS

• **Computing methodologies → Neural networks**;

KEYWORDS

graph data mining; few-shot classification; meta-learning; graph neural networks

ACM Reference Format:
Ning Ma[1,2,3], Jiajun Bu[1,2,3,*], Jieyu Yang[1,2,3], Zhen Zhang[1,2,3] and Chengwei Yao[1,2,3], Zhi Yu[1,2,3], Sheng Zhou[1,2,3], Xifeng Yan[4]. 2020. Adaptive-Step Graph Meta-Learner for Few-Shot Graph Classification. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411951

1 INTRODUCTION

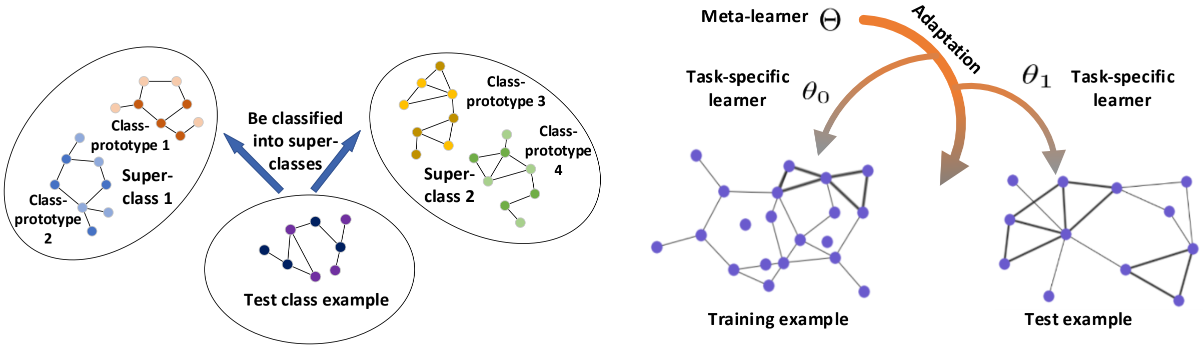
Many real-world networks can be formulated as graphs for modeling different relationships among nodes, such as social networks, chemical molecule structures and citation networks. Recently, there have been various attempts to extend the Convolutional Neural Network (CNNs) and pooling methods to graph-structured data. These methods are named as Graph Neural Networks (GNNs) and have been successfully applied to different graph related tasks containing graph classification, node classification and link prediction [55, 57]. Graph classification aims to extract accurate information from graph-structured data for classification. However, most existing GNNs based graph classification methods overlook that it's complicated and time consuming to collect or label the graph data. Learning with few labeled graph data is still a challenge for the practical applications of graph classification.

Few-shot learning, aiming to label the query data when given only a few labeled support data, is a natural way to alleviate the problem. There are many papers discussing few-shot learning with meta-learning [12], data augmentation [53] or regularization [50], but most of them don't consider the graph data. Furthermore, there have been several methods for few-shot node classification [20, 38, 56] and few-shot link prediction [10, 23, 32, 43], but they only focus on node-level embedding. Recently, Chauhan et al. [7] proposed few-shot graph classification based on graph spectral measures and got satisfactory performance. From the global structure of dataset, they *bridge* the test classes and training classes by assuming that the test classes belong to the same set of super-classes clustered from training classes. However, the above methods based on graph spectral measures might have some limitations for the following reasons: (1) the label spaces of training classes and test classes usually do not overlap in few-shot settings; (2) the bridging methods above may diminish the model to capture the local structure of test data.

[*]Corresponding Author.

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411951



(a) Graph spectral measures method [7].

(b) Our method.

Figure 1: Comparison of different methods. (a) The method from the global structure of the dataset. The most representative graph of each class is viewed as class-prototype graph. The super-classes are clustered from training classes. (b) Our method from the view of local structure. Θ is meta-learner's parameters and θ_1, θ_2 are task specific parameters derived from meta-learner's fast adaptation. The components in dotted boxes have similar triangle structure. We assume the similarity can be discovered by a well initialized meta-learner within a few adaptation steps.

From the perspective of the graph's local structure, we observe that the graphs of training classes and test classes have similar sub-structures. For example, different social networks usually have similar groups; different protein molecules usually have similar spike proteins. We assume these similarities can be discovered by a well initialized meta-learner within a few adaptation steps. Therefore, we consider fast adaptation by a meta-learner from learned graph classification tasks to new tasks. Figure 1 illustrates the existing method and our assumption.

Currently, GNNs have reliable ability to capture local structures over graphs by convolutional operations and pooling operations, but lack of fast adaptation mechanism when dealing with never seen graph classes. Inspired by Model Agnostic Meta-Learning (MAML, [12]), which has attracted great attention because of its fast adaptation mechanism, we leverage GNNs as graph embedding backbone and meta-learning as a training paradigm to rapidly capture task-specific knowledge in graph classification tasks and transfer them to new tasks.

However, directly applying MAML for fast adaptation is suboptimal due to the following reasons: (1) MAML requires painstaking hyperparameter searches to stabilize training and achieve high generalization [1]; (2) unlike images, graphs have arbitrary node size and sub-structure, which brings uncertainty for adaptation. There have some variants of MAML trying to overcome these problems by incorporating an online hyperparameter adaptation [4], reducing optimization difficulty [31] or increasing context parameters for adaptation [58], but they don't consider the structure of graph data. In this paper, we design a novel component named as adaptive step controller to learn optimal adaptation step for meta-learner to improve its learning robustness and generalization. The controller evaluates the meta-learner and decides when to stop adaptation by two kinds of inputs: (1) graphs' embedding quality, which is viewed as a meta-feature and indicated with Average Node Information (ANI, the average amount of node information in a batch of graphs); (2) meta-learner's training state, which is indicated with training loss of classification.

We formulate our framework as **A**daptive **S**tep MAML (AS-MAML). To the best of our knowledge, we are the first to consider the few-shot graph classification problem from the view of the graph's local structure and propose a fast adaptation mechanism on graphs via meta-learning. Our contributions are summarized as follows:

- We propose a novel GNNs based graph meta-learner, which captures the features efficiently of sub-structures on unseen graphs by fast adaptation mechanism.
- We design a novel controller for meta-learner. Driven by Reinforcement Learning (RL, [19]), the controller provide optimal adaptation step for the meta-learner via graph's embedding quality and training loss. Our ablation experiments show its effectiveness to improve learning robustness and generalization.
- We perform quantitative analysis and provide a generalization guarantee of key algorithms via a graph-dependent upper bound.
- We evaluate our framework's performance against different baselines on four graph datasets and achieve state-of-the-art performance in almost all the tasks. We also evaluate the transferability of popular graph embedding modules on our few-shot graph classification tasks.

2 RELATED WORKS

2.1 Graph Classification

In graph classification tasks, each full graph is assigned a class label. There exist several branches for graph classification. The first is graph kernel methods which design kernels for the substructures exploration and exploitation of graph data. The typical kernels include Shortest-path Kernel [5], Graphlet Kernel [40] and Weisfeiler-Lehman Kernel [39].

As the main branch in recent years, GNNs have been successfully applied to graph classification. GNNs focus on node representations, which are iteratively computed by message passing from the features of their neighbor nodes using a differentiable aggregation operation. GCN [21] proposed Graph Convolutional Neural Network (termed as GCN) and got satisfying results based on directly feature aggregation from neighborhood nodes. GAT [44] imported attention mechanism for graph convolutional operations. GraphSAGE [16] proposed an inductive framework that leverages node features to generate node embeddings efficiently for unseen nodes. In our framework, we use these classical methods to update nodes of graphs, while other methods like Graph Isomorphism Network (GIN) [46] are also applicable.

In the meantime, inspired by pooling in CNNs, a bunch of researchers concentrates on efficient pooling methods for accurate graph summary and computation efficiency. Beyond pooling layers in CNNs, graph pooling layers can enable GNNs to reason and get global representation from adjacent nodes. More and more evidence shows that graph pooling promotes the graph classification performance [9, 13, 25]. SAGPool [25] implemented self-attention pooling on graphs considering both node features and graph topology. EdgePool [9] implemented a localized and sparse pooling transform backed by the notion of edge contraction. Graph U-nets [13] implemented novel graph pooling and unpooling operations. Based on these operations, they developed a new model containing graph encoder and graph decoder and got satisfactory performance on graph classification tasks.

2.2 Few-Shot Learning and Meta-Learning

Few-shot classification aims to learn a model under the circumstances of low sample resources and is usually powered by meta-learning in recent years. Meta-learning was also known as learning to learn, with a meta-learner observing various task learning processes and summarizing meta-knowledge to accelerate the learning efficiency of new tasks. Baxter et al. [3] proposed a model to learn inductive bias from the perspective of bias learning, and they analytically showed that the number of examples required of each task decreases as the number of task rises.

Recent meta-learning related works can be classified into the following categories: optimization (or gradients) based methods and memory based methods. The optimization based methods aim to train a model to learn optimization [26, 34], learn a good initialization [12] for rapid adaptation, or train parameter generator for task-specific classifier [36]. Moreover, the memory based methods learn new tasks by reminiscence mechanism in virtue of physical memory [37].

Furthermore, almost all the previous few-shot learning methods are devised for image data, where images are prone to be represented in Euclidean space. Because we all have the idea that CNNs based models can perform efficient transfer in Euclidean space by feature reuse [33], in virtue of that different images usually share common edge features and corner features. However, graph data such as social networks, which are appropriate to be formed into non-Euclidean space instead of Euclidean space. Few-shot learning in non-Euclidean space is addressed in our work.

2.3 GNNs and its Generalization on Graph data

We have seen several works of few-shot node classification promoting performance via GNNs [15, 20, 27–29, 38, 47–49], but they just leverage the message passing mechanism of GNNs to enhance the performance on node classification, without involving the generalization of GNNs themselves and compatibility with graph classification task. For graph classification, Knyazev et al. [22] focus on the ability of attention GNNs to generalize to larger, more complex or noisy graphs. Lee et al. [24] imported a domain transfer method by transferring the intrinsic geometric information learned in the source domain to the target. Hu et al. [18] systematically studied the effectiveness of pre-training strategies on multiple graph datasets. Based on graph spectral measures, Chauhan et al. [7] proposed few-shot graph classification using the notion of super-graph by two steps: (1) they define the p-th Wasserstein distance to measure the spectral distance among graphs and select the most representative graph as prototype graph for each class; (2) by clustering the prototype graphs based on spectral distance, they clustered the prototype graph again into a super-graph consisting of super-classes. Therefore, they assume that the test classes belong to the same set of super-classes clustered from the training classes. We loosen the assumption and emphasize fast adaptation to boost few-shot graph classification.

3 PROBLEM SETUP

We form the few-shot problem as N-way-K-shot graph classification. Firstly, given graph data $\mathcal{G} = \{(G_1, \mathbf{y}_1), (G_2, \mathbf{y}_2), \cdots, (G_n, \mathbf{y}_n)\}$, where $G_i = (\mathcal{V}_i, \mathcal{E}_i, \mathbf{X}_i)$. We use n_i to denote the number of node set $\mathcal{V}_i$. So each graph G_i has an adjacent matrix $\mathbf{A}_i \in \mathbb{R}^{n_i \times n_i}$, which is constructed from an edge set $\mathcal{E}_i$, and a node attribute matrix $\mathbf{X}_i \in \mathbb{R}^{n_i \times d}$, where d is the dimension of node attribute. Secondly, according to label $\mathbf{y}$, we split $\mathcal{G}$ into $\{(\mathcal{G}^{train}, \mathbf{y}^{train})\}$ and $\{(\mathcal{G}^{test}, \mathbf{y}^{test})\}$ as training set and test set respectively. Notice that $\mathbf{y}^{train}$ and $\mathbf{y}^{test}$ must have no common classes. We use episodic training method, which means at the training stage we sample a task $\mathcal{T}$ each time, and each task contains support data $D^{train}_{sup} = \{(G^{train}_i, \mathbf{y}^{train}_i)\}^{s}_{i=1}$ and query data $D^{train}_{que} = \{(G^{train}_i, \mathbf{y}^{train}_i)\}^{q}_{i=1}$, where s and q are the number of support data and query data respectively. Given labeled support data, our goal is to predict the labels of query data. Please note that in a single task, support data and query data share the same class space. If $s = N \times K$, which means that support data contain N classes and K labeled samples per class, we name the problem as N-way-K-shot graph classification. At the test stage when performing classification tasks on unseen classes, we firstly fine tune the meta-learner on the support data of test classes $D^{test}_{sup} = \{(G^{test}_i, \mathbf{y}^{test}_i)\}^{s}_{i=1}$, then we report classification performance on $D^{test}_{que} = \{(G^{test}_i, \mathbf{y}^{test}_i)\}^{q}_{i=1}$.

4 PROPOSED FRAMEWORK

Overall, our few-shot graph classification framework consists of GNNs based meta-learner and a step controller to decide the adaptation steps of meta-learner. We use MAML to implement a fast adaptation mechanism for meta-learner because of its model agnostic property. Du et al. [10] proposed an RL based step controller to guide meta-learner for link prediction. We argue that classification

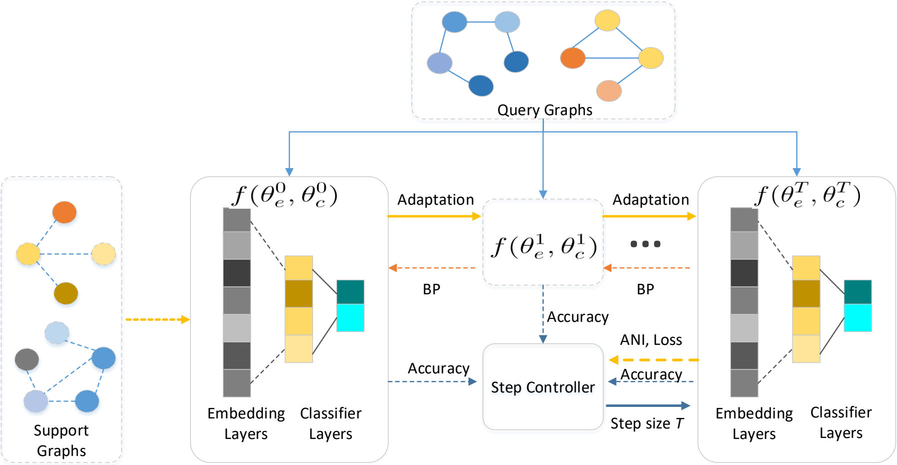


Figure 2: Diagram of the AS-MAML framework's learning process in a single episode on the 2-way-1-shot graph classification task. The yellow arrows show meta-learner's T step adaptations on support graphs. The blue dash arrows show T step evaluations (Accuracies) on the query graphs. The orange dash arrows show the backpropagation (BP) according to T-th loss on query graphs. The step controller receives ANIs and classification losses on support graphs of each step. After that, the controller outputs the adaptation step T. Finally, the controller receives accuracies on query graphs as rewards and updates its own parameters.

loss is suboptimal to be viewed as rewards for overcoming overfitting. Therefore, we adopt a novel step controller to accelerate training and overcome overfitting. Our step controller is also driven by RL but learns the optimal adaptation step by using ANIs and losses as inputs and classification accuracy as rewards. Figure 2 demonstrates the training process of our framework.

4.1 Graph Embedding Backbone

We explain our proposed framework with typical graph convolutional modules and pooling modules as embedding backbone, due to that novel graph convolutional modules or pooling modules are out of concern for this paper. The first step to represent a graph is to embed the nodes it contains. We investigate several embedding methods such as GCN, GAT, GraphSAGE and GIN. Here we focus on GraphSAGE as following reasons: (1) GraphSAGE has more flexible aggregators in few-shot learning scenarios; (2) Errica et at. [11] set GraphSAGE as a strong baseline when compared to GIN for the graph classification task. Hence we use mean aggregator of GraphSAGE as follows:

$$\mathbf{h}_v^l = \sigma\left(\mathbf{W} \cdot \text{mean}\left(\left\{\mathbf{h}_v^{l-1}\right\} \cup \left\{\mathbf{h}_u^{l-1}, \forall u \in \mathcal{N}(v)\right\}\right)\right), \qquad (1)$$

where $\mathbf{h}_v^l$ is the l-th layer representation of node v, σ is the sigmoid function, $\mathbf{W}$ is the parameters and $\mathcal{N}(v)$ contains the neighborhoods of v. Please note that this mean aggregator just belongs to the group of typical aggregators we use in experiments. We will provide concrete analysis for other aggregators in Section 5 and Section 6.4.

After that, we discuss existing pooling operations. Under the circumstances of few-shot learning, the meta-learner urgently needs a flexible pooling strategy with learning capability to strengthen its generalization. Here, we focus on self-attention pooling (SAGPool) [25] as our pooling layer thanks to its flexible attention parameters. The main step of SAGPool is to calculate the attention score matrix of graph G_i as follows:

$$\mathbf{S}_i = \sigma\left(\tilde{\mathbf{D}}_i^{-\frac{1}{2}}\tilde{\mathbf{A}}_i\tilde{\mathbf{D}}_i^{-\frac{1}{2}}\mathbf{X}_i\Theta_{att}\right), \qquad (2)$$

where the $\mathbf{S}_i \in \mathbb{R}^{n_i \times 1}$ indicates the self-attention score, n_i is node number of the graph. σ is the activation function (e.g., tanh), $\tilde{\mathbf{A}}_i \in \mathbb{R}^{n_i \times n_i}$ is the adjacency matrix with self-connections, $\tilde{\mathbf{D}}_i \in \mathbb{R}^{n_i \times n_i}$ is the diagonal degree matrix of $\tilde{\mathbf{A}}_i$, $\mathbf{X}_i \in \mathbb{R}^{n_i \times d}$ is n input features with dimension d, and $\Theta_{att} \in \mathbb{R}^{d \times 1}$ is the learnable parameters of pooling layer. Based on the attention score, we select top $c < n_i$ nodes that have larger scores with keeping their origin edges unchanged.

To get fixed representation dimension for each graph, we need Read-Out operation to form each graph embedding vector into identical dimension. Following Zhang et al. [54], we use the concatenation of mean-pooling and max-pooling for each level of graph embeddings of G_i as follows:

$$\mathbf{r}_i^l = \mathcal{R}\left(\mathbf{H}_i^l\right) = \sigma\left(\frac{1}{n_i^l}\sum_{p=1}\mathbf{H}_i^l(p,:) \| \max_{q=1}^{d}\mathbf{H}_i^l(:,q)\right), \qquad (3)$$

where $\mathbf{r}_i^l \in \mathbb{R}^{2d}$ is the l-th layer embedding, n_i^l is the node number in l-th layer, $\mathbf{H}_i^l$ denotes l-th layer hidden representation matrix , $\|$ is concatenation operation, p and q are row number and column number respectively, d is feature dimension, and σ is the activation function (e.g., Rectified Linear Unit, ReLU [8]).

Algorithm 1 Training Stage of AS-MAML

```
Input: Task distribution $p(\mathcal{T})$ over $\{(G^{train}, \mathbf{y}^{train})\}$
Parameter: Graph embedding parameters $\theta_e$, classifier parameters $\theta_c$, step controller parameters $\theta_s$, learning rate $\alpha_1, \alpha_2, \alpha_3$
Output: The trained parameters $\theta_e, \theta_c, \theta_s$
 1: Randomly initialize $\theta_e$, $\theta_c$, $\theta_s$
 2: while not convergence do
 3:     Sample task $\mathcal{T}_i$ with support graphs $D^{train}_{sup}$ and query graphs $D^{train}_{que}$
 4:     Get adaptation step $T$ via Equation 8
 5:     Set fast adaptation parameters: $\theta' = \theta = \{\theta_e, \theta_c\}$
 6:     for $t = 0 \rightarrow T$ do
 7:         Evaluate $\nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'})$ on $D^{train}_{sup}$ by classification loss $\mathbf{L}^{(t)}$.
 8:         Update $\theta'$ : $\theta' \leftarrow \theta' - \alpha_1 \nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'})$
 9:         Calculate ANI $\mathbf{M}^{(t)}$ via Equation 6
10:         Calculate stop probability $p(t)$ via Equation 7
11:         Calculate reward $Q^{(t)}$ on $D^{train}_{que}$ by Equation 9
12:     end for
13:     $\theta \leftarrow \theta - \alpha_2 \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'})$ on $D^{train}_{que}$
14:     for $t = 0 \rightarrow T$ do
15:         $\theta_s \leftarrow \theta_s + \alpha_3 Q^{(t)} \nabla_{\theta_s} \ln p(t)$
16:     end for
17: end while
```

Following the graph embedding backbone, we compute the final graph embedding of G_i as

$$\mathbf{z}_i = \mathbf{r}_i^1 + \mathbf{r}_i^2 + \cdots + \mathbf{r}_i^L \tag{4}$$

and put it into Multi-Layer Perceptron (MLP) classifier to perform classification using cross-entropy loss.

4.2 Meta-Learner for Fast Adaptation

We use θ_e and θ_c to denote the parameters of graph embedding modules and MLP classifier respectively, where θ_e contains the parameters of node embedding layers and pooling layers. To achieve the fast adaptation of θ_e and θ_c, we put them into a nested loop framework to create a GNNs based meta-learner. Specifically, our meta-learner is optimized from two procedures. One of the procedures is called the outer loop aiming to get optimal initialization for new classification tasks, and one is called the inner loop to implement fast adaptation based on a suitable initialization. Algorithm 1 elaborates on how to train a graph meta-learner at the training state. First, we sample support data D^{train}_{sup} and query data D^{train}_{que} in an episode. Then we perform adaptation operation by updating θ_e and θ_c for T steps on D^{train}_{sup}. Lines 7 to 8 in Algorithm 1 demonstrate the adaptation of meta-learner. After adaptation, demonstrated by line 13, we use the losses on D^{train}_{que} to perform backpropagation and update θ_e as well as θ_c. Similarly, at the test stage, the meta-learner will perform adaptation on labeled support graphs D^{test}_{sup} and predict the label of query graphs D^{test}_{que}.

4.3 Adaptation Controller

Finding optimal combinations of learning rates and step size is difficult for MAML [1]. Besides, arbitrary graph size and structure bring difficulty for ascertaining optimal step size manually. As an empirical solution to alleviate these problems, we design an RL based controller to decide optimal step size for the adaptation of meta-learner when given other hyper-parameters. Therefore, our controller must roughly know when to stop adaptation according to the embedding quality and training state (denoted by loss). We focus on Average Node Information (ANI) to indicate the embedding quality. Intuitively, if a node can be well reconstructed by its neighborhoods, it has less information for the graph classification. Similarly, the rising of batch graphs' ANI indicates that the pooling module has learned to select the most informative nodes. Hou et al. [17] proposed similar concept called *Feature Smoothness* measuring node information over graphs. Here we adopt another practical method defined by [54], where they compute node information as the Manhattan distance between the node representation itself and the one constructed from its neighbors. Inspired by their work, we define the ANI of a single graph G_i as follows:

$$ANI_i^l = \frac{1}{n_i^l} \sum_{j=1} \left\| \left[\left(\mathbf{I}_i^l - \left(\mathbf{D}_i^l \right)^{-1} \mathbf{A}_i^l \right) \mathbf{H}_i^l \right]_j \right\|_1 , \tag{5}$$

where l denotes the embedding layer of the graph, n_i^l denotes the number of node, j denotes the row index of matrix or j-th node, $\|\cdot\|_1$ denotes the L1 norm of row vector, $\mathbf{A}_i^l$ denotes the adjacency matrix, $\mathbf{D}_i^l$ is the degree matrix of $\mathbf{A}_i^l$, $\mathbf{H}_i^l$ denotes l-th layer hidden representation matrix. In our work, we only use the last layer of graph embedding (i.e., $\mathbf{H}_i^L$). And unless specifically stated, we use scalar value ANI to denote the average node information of the batch graphs:

$$ANI = 1/n * \sum_{i=1}^{n} ANI_i^L, \tag{6}$$

where n denotes the number of batch graphs, L denotes the L-th layer of graph embedding.

Next we set the number of initial step as T_i, the ANIs in T_i steps as $\mathbf{M} \in \mathbb{R}^{T_i \times 1}$ and denote classification losses as $\mathbf{L} \in \mathbb{R}^{T_i \times 1}$. Then we compute stop probability $p^{(t)}$ at step t as follows:

$$\boldsymbol{h}^{(t)} = \text{LSTM}\left(\left[\mathbf{L}^{(t)}, \mathbf{M}^{(t)} \right], \boldsymbol{h}^{(t-1)} \right), p^{(t)} = \sigma\left(\mathbf{W}\mathbf{h}^{(t)} + \mathbf{b} \right), \tag{7}$$

where $\mathbf{W}$ and $\mathbf{b}$ are the parameters of a MLP module, σ is sigmoid function and $h^{(t)}$ is the output of LSTM module. Note that the adaptation will not stop until T_i steps at current task regardless of $p^{(t)}$. We set $p^{(t)}$ as a prior for the next task:

$$T_{i+1} = \left\lfloor \frac{1}{p^{(T_i)}} \right\rfloor, \tag{8}$$

where $\lfloor\rfloor$ is round down operation. We can observe that we compute T of next task according to the ANIs and losses, where both of them are produced by current task. The reason behind it is that if we stop adaptation according to $p^{(t)}$ in current task, it will output larger variance of step and bring instability for optimization. Besides, we get controller's rewards as following:

$$Q^{(t)} = \sum_{t=1}^{T} r^{(t)} = \sum_{t=1}^{T} (e_T - e_t - \eta * t), \tag{9}$$

where T is total steps and e_t is the classification accuracy on query data at step t, and $\eta * t$ denotes the penalty item. Then we update our controller by policy gradients, which is a typical method in RL:

$$\theta_s = \theta_s + \alpha_3 Q^{(t)} \nabla_{\theta_s} \ln p(t), \tag{10}$$

where $\ln p(t)$ is the log of stop probability p(t) ·, ∇_{θ_s} is the gradients over θ_s and α_3 is learning rate.

5 GRAPH STRUCTURE-DEPENDENT BOUND FOR META-LEARNING

In this section, we theoretically analyze the proposed framework with unnecessary details omitted. Driven by graph data, we will build a generalization error bound in the meta-learning scenario considering the locality of graph structure (e.g., the degrees of nodes in a graph). Before getting closer to analysis, we first give the key results: (1) The bound is dependent on the number of graphs in the support example of test classes and the number of graphs of training classes. (2) The bound is dependent on the locality structure of graphs between training classes and test classes. The first result is common in meta-learning scenarios, while the second result is derived from the structure of graphs.

From the perspective of representation learning and probability distribution, our framework try to minimize the distance of latent distributions between $\mathcal{G}^{train}$ and $\mathcal{G}^{test}$ directly. Based on this premise, we use the integral probability metric (IPM, [30]) as a general distance metric to produce the upper bound. IPM has been employed in the analysis of transfer learning [51], non-parametric estimation [42] and generative models [52]. Generally, IPM can be formalized as:

$$\gamma_{\mathcal{H}}(\mathbb{P}, \mathbb{Q}) := \sup_{h \in \mathcal{H}} \left| \mathbb{E}_{\mathbb{P}} h(z) - \mathbb{E}_{\mathbb{Q}} h(z) \right|, \tag{11}$$

where $\mathbb{P}$ and $\mathbb{Q}$ denote two different probability distributions, respectively, $\mathcal{H}$ denotes the collection of real-valued functions (e.g., square loss function and margin loss function).

We set the empirical distribution of $\mathcal{G}^{train}$ as $\hat{\mathbb{P}}$. With a slight abuse of notation, in a single task at the test stage, we set the empirical distribution of support graph D_{sup}^{test} as $\hat{\mathbb{Q}}$, and set the expected distribution query data D_{que}^{test} as $\mathbb{Q}$. Then in the adaptation process at test stage, we actually want to minimize the following bound:

$$\gamma_{\mathcal{H}}(\hat{\mathbb{P}}, \mathbb{Q}) = \sup_{h \in \mathcal{H}} \left| \mathbb{E}_{\hat{\mathbb{P}}} h(G) - \mathbb{E}_{\mathbb{Q}} h(G) \right|, \tag{12}$$

where $h(G)$ is the classification loss of meta-learner after the adaptation on D_{sup}^{test}. Then with the help of IPM, $\gamma_{\mathcal{H}}(\hat{\mathbb{P}}, \mathbb{Q})$can be bounded by following theorem [6]:

Theorem 1. *Let $\mathcal{H}$ denote a class of functions whose members map from G_i to [a, b], and suppose that the training data test data are independent, and that the data instances of each are i.i.d. within a sample. Let $\epsilon > 0$. Then with probability at least $1 - \epsilon$ over the draws of the training and query samples,*

$$\gamma_{\mathcal{H}} \left(\hat{\mathbb{P}}, \mathbb{Q}\right) \le \gamma_{\mathcal{H}} \left(\hat{\mathbb{P}}, \hat{\mathbb{Q}}\right) + 2\mathcal{R}\left(\mathcal{H} | \{G_i\}_{i=1}^{m}\right) + 3\sqrt{\frac{(b-a)^2 \log(2/\epsilon)}{2m}}, \tag{13}$$

where m is the number of support graphs D_{sup}^{test}, $\mathcal{R}\left(\mathcal{H} | \{G_i\}_{i=1}^{m}\right)$ denotes the empirical Rademacher complexity [2] of the function class $\mathcal{H}$ w.r.t. the support graphs.

Now we provide the proof of Theorem 1 referencing Cai et al. [6]. First, IPM is interrelated with *uniform deviation with empirical Rademacher complexity* [2]:

Theorem 2. *Let $\mathcal{H}$ denote a class of functions whose members map from z_i to [a, b], and suppose that $\{z_i\}_{i=1}^{m}$ is sampled from a i.i.d distribution $\mathbb{P}$. Then for $\epsilon > 0$. Then with probability at least $1 - \epsilon$ over the sample,*

$$\sup_{h \in \mathcal{H}} \left| \mathbb{E}_{\hat{\mathbb{P}}} h(z) - \mathbb{E}_{\mathbb{P}} h(z) \right| \le 2\mathcal{R}\left(\mathcal{H} | \{z_i\}_{i=1}^{m}\right) + 3\sqrt{\frac{(b-a)^2 \log(2/\epsilon)}{2m}}, \tag{14}$$

where $\hat{\mathbb{P}}$ represents the empirical distribution of the sample, $\mathcal{R}\left(\mathcal{H} | \{z_i\}_{i=1}^{m}\right)$ denotes the empirical Rademacher complexity of the function class $\mathcal{H}$ w.r.t. the sample.

Proof. Then according to Cai et al. [6], we can derive the Equation 12 as:

$$\gamma_{\mathcal{H}}(\hat{\mathbb{P}}, \mathbb{Q}) \le \gamma_{\mathcal{H}}\left(\hat{\mathbb{P}}, \hat{\mathbb{Q}}\right) + \sup_{h \in \mathcal{H}} \left| \mathbb{E}_{\hat{\mathbb{Q}}} h(G) - \mathbb{E}_{\mathbb{Q}} h(G) \right| \tag{15}$$

When the last term of Formula 15 substituted by Theorem 14, Theorem 1 is proved. □

After giving the proof of Theorem 1, $\gamma_{\mathcal{H}}\left(\hat{\mathbb{P}}, \hat{\mathbb{Q}}\right)$ can be optimized by the adaptation of meta-learner on the test stage. As the final step, we focus on the evaluation of $\mathcal{R}\left(\mathcal{H} | \{G_i\}_{i=1}^{m}\right)$. The diversity of graph structure brings difficulty for directly calculating this term. We adopt the method proposed by K. Garg et al. [14], who gave an upper bound for $\mathcal{R}\left(\mathcal{H} | \{G_i\}_{i=1}^{m}\right)$, by transforming each graph to the corresponding collection of local computation trees. Because when updating every node embedding, the node to be updated can be viewed as a root node of a tree, with its neighbor nodes as children. Given this hypothesis, they derived a strict upper bound mainly based on the degree of nodes, where the degree was used to denote the local complexity of the graph. Furthermore, the local complexity of the graph is made full use by our model. In the experiment of Section 6.5, our model performs better over datasets which have clear local structure.

6 EXPERIMENTS

In the experiments, we focus on two aspects: (1)How does the framework perform on few-shot graph classification tasks? (2) How does the controller work when training meta-learner? In this section, we will introduce experiment datasets, comparison with baselines and details of implementation. Finally, we demonstrate the effectiveness of key modules by ablation study and detail analysis.

Datasets	$\|G\|$	Avg.$\|\mathcal{V}\|$	Avg.$\|\mathcal{E}\|$	C_0	C_1	C_2
COIL-DEL	3900	21.54	54.24	60	16	20
Graph-R52	8214	30.92	165.78	18	5	5
Letter-High	2250	4.67	4.50	11	0	4
TRIANGLES	45000	20.85	35.50	7	0	3

Table 1: Statistics of datasets. For each dataset, we show total graph number $|G|$, average node number Avg.$|\mathcal{V}|$, Average edge number Avg.$|\mathcal{E}|$ and class number for training (C_0), validation (C_1) and test (C_2).

6.1 Datasets

We select four public graph datasets including COIL-DEL, R52, Letter-High and TRIANGLES. These datasets are publicly available [1] [2]. The statistics are summarized in Table 1. The visualization of each datasets are shown in Figure 3. In the work proposed by [7] for few-shot graph classification, they focus on two datasets containing Letter-High and TRIANGLES. We use two additional datasets where Graph-R52 was built from text classification dataset and COIL-DEL was built from images.

COIL-DEL. COIL-DEL is built on images, and each graph is constructed by applying corner detection and Delaunay triangulation to corresponding image [35].

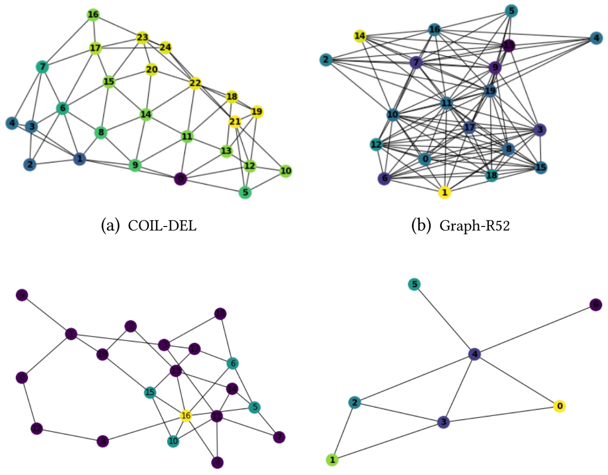
R52. R52 is a text dataset in which each text is viewed as a graph. We transformed it into a graph dataset as follows: if two words appear together in a specified sliding window, they have an undirected edge in the graph. We keep classes with more than 20 samples and finally get 28 classes. We name the new dataset as Graph-R52 for clarity.

Letter-High. Each graph represents distorted letter prototype drawings with representing lines by undirected edges and ending points of lines by nodes [35]. More specifically, Letter-High contains 15 classes from English alphabets: A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z.

TRIANGLES. The dataset was proposed for the task of triangle counting, where the model is required to give the number of triangles of each graph. TRIANGLES contains 10 different graph classes numbered from 1 to 10 corresponding to the number of triangles in graphs of each class. In our experiments, we use the partition of [7], where they remove oversize graph samples so the total sample size of TRIANGLES is reduced from 45000 to 2000.

6.2 Baselines

We adopt four groups of baselines made up of Graph Kernel, Finetuning, GNNs-Prototypical-Classifier (GNNs-Pro) and Graph Spectral Measures (GSM) [7] . For Graph Kernel baselines, we perform N-way-K-shot graph classification over the test set directly because there are no parameters to transfer. The baselines of the last three groups train a GNNs based graph classifier by performing classification over C_0 training classes (see Table 1). On the test stage, they perform N-way-K-Shot classification.



(a) COIL-DEL

(b) Graph-R52

(c) TRIANGLES

(d) Letter-High

Figure 3: Visualization of typical instances of the used datasets, where their different graph size and graph structures bring challenges for graph classification models.

Graph Kernel. This group of methods firstly measure the similarity between labeled support data and query data on the test stage. After that, the similarity matrix was put into a Prototypical Classifier, which has none of the parameters [41], to get predicted labels of query data. We choose typical graph kernel algorithms including Shortest-path Kernel (SP) [5], Graphlet Kernel [40] and Weisfeiler-Lehman Kernel (WL) [39].

Finetuning. In this baseline, we train a naive graph classifier consisting of GraphSAGE, SAGPool and MLP classifier. On the test stage, we change the output dimension of the last layer of the classifier and fine-tune (re-train) the layer's parameters, while keeping other modules unchanged.

GNNs-Pro. We train a graph classifier following Finetuning. On the test stage, we replace the MLP classifier with Prototypical Classifier. We choose GCN [21], GraphSAGE [16] and GAT [44] as graph convolutional modules, and Self-attention Pooling (SAGPool) [25], TopK Pooling (TopKPool) [13] and Edge Pooling (EdgePool) [9] as graph pooling modules.

GSM. Chauhan et al. [7] proposed the GSM based method customized for few-shot graph classification. On the training stage, they compute prototype graphs from each class, then they cluster the prototype graphs to produce super-classes. After that, they predict the origin class and super-class of each graph. On the test stage, they only update the classifier based on the classification of origin classes.

6.3 Experimental Details

To ensure a fair comparison, we use three convolutional layers followed by corresponding pooling layers for the GNNs based baselines and our proposed framework. We set the same node dimension as 128 for all GNNs based baselines. For the adaptation step, we

[1] https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets
[2] https://www.cs.umb.edu/ smimarog/textmining/datasets/

Categories	Baselines	COIL-DEL		Graph-R52	
		5-way-5-shot	5-way-10-shot	2-way-5-shot	2-way-10-shot
Kernels	GRAPHLET	47.47 ± 1.06	49.04 ± 0.98	56.52 ± 1.46	57.16 ± 1.47
	SP	38.33 ± 0.62	42.18 ± 0.69	74.38 ± 1.50	76.96 ± 1.34
	WL	43.05 ± 1.25	52.28 ± 1.47	**76.90 ± 1.48**	75.91 ± 1.46
Finetuning	finetuning	68.21 ± 1.29	72.38 ± 1.40	71.87 ± 2.04	72.39 ± 1.88
GNNs-Pro	GCN, TopKPool	78.01 ± 1.83	78.98 ± 1.53	69.98 ± 1.53	70.19 ± 1.37
	GCN, EdgePool	76.21 ± 1.54	79.43 ± 1.58	67.24 ± 1.34	67.72 ± 1.59
	GCN, SAGPool	76.58 ± 1.19	79.16 ± 1.06	69.88 ± 1.40	70.46 ± 1.47
	GraphSAGE, TopKPool	69.80 ± 1.25	74.18 ± 1.73	70.43 ± 1.76	70.52 ± 1.83
	GraphSAGE, EdgePool	80.08 ± 1.26	80.96 ± 1.26	68.13 ± 1.59	70.72 ± 1.58
	GraphSAGE, SAGPool	79.30 ± 1.12	80.91 ± 1.62	68.10 ± 1.40	70.49 ± 1.32
	GAT, TopKPool	76.37 ± 1.10	77.29 ± 1.40	71.99 ± 1.51	73.31 ± 1.44
	GAT, EdgePool	81.00 ± 1.22	83.57 ± 0.99	66.49 ± 1.32	70.49 ± 1.17
	GAT, SAGPool	72.54 ± 1.07	73.99 ± 1.00	67.78 ± 1.52	74.10 ± 1.57
Ours	AS-MAML (wo/AS)	79.54 ± 1.48	81.24 ± 1.27	74.12 ± 1.39	76.05 ± 1.17
	AS-MAML (w/AS)	**81.55 ± 1.39**	**84.75 ± 1.30**	75.33 ± 1.19	**78.33 ± 1.17**

Table 2: Accuracies with a standard deviation of baseline methods and our framework. We tested 200 and 500 N-way-K-shot tasks on COIL-DEL and Graph-R52, respectively. The bold black numbers denote the best results we get, and the blue numbers denote the second best results. AS-MAML (wo/AS) denotes our framework without Adaptive Step (AS) which is controlled by our step controller, and AS-MAML (w/AS) denotes the whole framework we proposed.

Methods	Shots	Datasets	
		TRIANGLES	Letter-High
GSM	5-shot	71.40 ± 4.34	69.91 ± 5.90
	10-shot	75.60 ± 3.67	73.28 ± 3.46
Ours	5-shot	86.47 ± 0.74	76.29 ± 0.89
	10-shot	87.26 ± 0.69	77.87 ± 0.75

Table 3: Accuracies evaluated from GSM and AS-MAML we proposed. For AS-MAML, we test 200 N-way-K-shot tasks on both datasets. For GSM, we use the best results in their paper.

set the minimum and maximum step by 4 and 15. We implement GNNs based baselines and our framework with PyTorch Geometric (PyG [3]) and graph kernel baselines based on GraKel [4]. We use SGD optimizer with 1e-5 for weight decay and versatile learning rates 0.0001, 0.001, 0.0001 for α_1, α_2 and α_3, respectively.

6.4 Comparison with Graph Kernel, Finetuning and GNNs-Pro

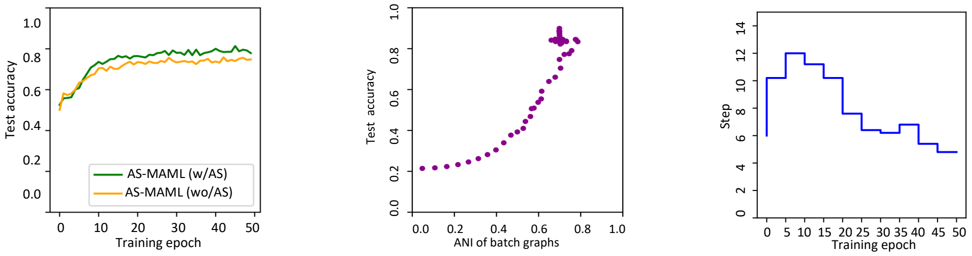
To evaluate the performance of our framework, we performed 5-way-5-shot and 5-way-10-shot graph classification on the COIL-DEL dataset. On the Graph-R52 dataset, we performed 2-way-5-shot and 2-way-10-shot graph classification. The results are reported in Table 2. Our framework utilizes GraphSAGE and SAGPool as graph embedding backbone. So firstly we compare our framework with the finetuning baseline built on GraphSAGE and SAGPool. We found our framework is superior to the finetuning baseline with a large margin, which indicates that the meta-leaner works well with a fast adaptation mechanism. Moreover, under the 5-way-10-shot setting in the GraphSAGE-SAGPool baseline, our framework achieves about 3.84% improvement on the COIL-DEL dataset.

Surprisingly, traditional graph kernel based baselines achieve competitive performance on the Graph-R52 dataset. The reasons are two-fold: (1) our graphs from texts contain many well defined sub-graphs built by text themes and their neighbor words, and this pattern gives graph kernels a favorable position; (2) the parameters of kernel methods are far less than GNNs based methods. So they are not prone to be overfitting, which is GNNs' common problem for the few-shot task. However, finding an appropriate kernel is difficult (e.g., From Table 2, SP behaves badly compared to WL and GRAPHLET on the COIL-DEL dataset).

6.5 Comparison with GSM Based Method

The GSM based method proposed by Chauhan et al. [7] did not adopt the episodic training paradigm, which is a key idea in our paper, so the method is inappropriate to be trained by N-way-K-shot graph classification on COIL-DEL and Graph-R52 dataset. For a fair comparison, we evaluate our framework on TRIANGLES and Letter-High, which are typical datasets used in their paper. As their partition, we randomly split out 20% examples from the training set to perform validation. Following their test configuration, we perform 3-way-K-shot and 4-way-K-shot classification on TRIANGLES and Letter-High respectively. The comparison of performances is shown in Table 3. From the table, we conclude that our framework outperforms theirs with a large margin. The reason behind it is that they assume the test classes belong to the same set of super-classes built from the training classes. However, the label spaces of training classes and test classes usually do not overlap in few-shot

[3] https://github.com/rusty1s/pytorch_geometric
[4] https://github.com/ysig/GraKeL



(a) Test accuracies

(b) ANI & training accuracies

(c) Adaptation steps

Figure 4: Illustrations of the learning process under the 5-way-10-shot setting on the COIL-DEL dataset. (a) Test accuracies calculated from AS-MAML (wo/AS) and AS-MAML (w/AS) respectively. The adaptation step of AS-MAML (wo/AS) is 6, and other hyper-parameters are same as AS-MAML (w/AS). The initial values are reported after 0-th training epoch. (b) The normalized ANIs and training accuracies in the first 50 epochs. Both of them are extracted from support graphs of the training set. (c) The variations of the adaptation step on the training stage. the value at epoch 0 is the initial adaptation step, and then we calculate an average for every 5 epochs.

settings. We observe that the graphs of training classes and test classes have similar sub-structures, which can be discovered by a well initialized meta-learner within a few adaptation steps. As mentioned before, different classes in TRIANGLES have similar triangle structure. Therefore our framework gets the most obvious improvement on this dataset.

6.6 Ablation Study and Detail Analysis

In this section, we show the effect of the controller module by ablation study. First of all, without the adaptive step (AS), we evaluate the performance of our framework by just putting GraphSAGE, SAGPool into meta-learner. From Table 2, we found that under the 2-way-10-shot setting on Graph-R52, AS-MAML (wo/AS) brings about 3.06% improvement compared with the finetuning baseline. Furthermore, the step controller brings about 2.28% improvement under the 2-way-10-shot setting on Graph-R52 and 3.51% improvement under the 5-way-5-shot setting on COIL-DEL.

We did a deeper analysis for ANI and give more details of the step controller module under the 5-way-10-shot setting on the COIL-DEL dataset. Figure 4(a) shows the effect on the test set after adding the adaptive step (AS). The scatter diagram (Figure 4(b)) shows that ANIs have a positive correlation with classification accuracies, which means larger ANI indicates better graph embedding for the MLP classifier module. The advantage of ANI against classification accuracy is that a larger ANI implies more discriminative graph embedding modules, while a better classification accuracy may mean that the MLP classifier module is overfitted on poor graph embedding modules. Finally, Figure 4(c) shows the variations of the adaptation step produced by the controller. At the beginning, the controller receives larger losses and smaller ANIs, so it gives more adaptation steps to meta-learner for encouraging exploration. When the meta-learner has been trained well, the controller receives smaller losses and larger ANIs, so it outputs smaller step size to alleviate overfitting.

7 CONCLUSION AND FUTURE WORKS

Modeling real-world data into graphs is getting more attention in recent years. In this paper, we focus on the few-shot graph classification and propose a novel framework named AS-MAML. To control the meta-learner's adaptation step, we proposed a novel step controller in a RL way by using ANI to demonstrate embedding quality. Beyond that, ANI is calculated by an unsupervised way like estimating Mutual Information on graphs [45]. Exploring and utilizing them for graph representation learning is an interesting future work. Moreover, we expect better graph embedding methods to improve the performance of our framework, including GIN and its variants. We also expect our work can be expanded to more challenging graph classification tasks like skeleton based action recognition, protein classification and subgraph analysis of social networks.

ACKNOWLEDGMENTS

This work is supported by Alibaba-Zhejiang University Joint Institute of Frontier Technologies, The National Key R&D Program of China (No. 2018YFC2002603, 2018YFB1403202), Zhejiang Provincial Natural Science Foundation of China (No. LZ13F020001), the National Natural Science Foundation of China (No. 61972349, 61173185, 61173186) and the National Key Technology R&D Program of China (No. 2012BAI34B01, 2014BAK15B02).

REFERENCES

[1] Antreas Antoniou, Harrison Edwards, and Amos Storkey. 2019. How to train your MAML. In *ICLR*.
[2] Peter L. Bartlett and Shahar Mendelson. 2003. Rademacher and Gaussian Complexities: Risk Bounds and Structural Results. *J. Mach. Learn. Res.* 3, null (March 2003), 463âĂŞ482.
[3] Jonathan Baxter. 2000. A Model of Inductive Bias Learning. *J. Artif. Int. Res.* 12, 1 (March 2000), 149âĂŞ198.
[4] Harkirat Singh Behl, AtÄślÄśm GÃijneÅ§ Baydin, and Philip H. S. Torr. 2019. Alpha MAML: Adaptive Model-Agnostic Meta-Learning. arXiv:cs.LG/1905.07435
[5] K. M. Borgwardt and H. P. Kriegel. 2005. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05).* 8 pp.–.
[6] Diana Cai, Rishit Sheth, Lester Mackey, and Nicolo Fusi. 2020. Weighted Meta-Learning. arXiv:stat.ML/2003.09465

[7] Jatin Chauhan, Deepak Nathani, and Manohar Kaul. 2020. Few-Shot Learning on Graphs via Super-Classes Based on Graph Spectral Measures. In *ICLR*.

[8] G. E. Dahl, T. N. Sainath, and G. E. Hinton. 2013. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 8609–8613.

[9] Frederik Diehl. 2019. Edge Contraction Pooling for Graph Neural Networks. *arXiv preprint arXiv:1905.10990* (2019).

[10] Zhengxiao Du, Xiaowei Wang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Sequential Scenario-Specific Meta Learner for Online Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD âĂŹ19)*. Association for Computing Machinery, New York, NY, USA, 2895âĂŞ2904. https://doi.org/10.1145/3292500.3330726

[11] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. 2020. A Fair Comparison of Graph Neural Networks for Graph Classification. In *ICLR*.

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICMLâĂŹ17)*. JMLR.org, 1126âĂŞ1135.

[13] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. In *ICML (PMLR)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. Long Beach, California, USA, 2083–2092.

[14] Vikas K. Garg, Stefanie Jegelka, and Tommi Jaakkola. 2020. Generalization and Representational Limits of Graph Neural Networks. arXiv:cs.LG/2002.06157

[15] Spyros Gidaris and Nikos Komodakis. 2019. Generating Classification Weights With GNN Denoising Autoencoders for Few-Shot Learning. In *CVPR*.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 1024–1034.

[17] Yifan Hou, Jian Zhang, James Cheng, Kaili Ma, Richard T. B. Ma, Hongzhi Chen, and Ming-Chang Yang. 2020. Measuring and Improving the Use of Graph Information in Graph Neural Networks. In *ICLR*. https://openreview.net/forum?id=rkeIIkHKvS

[18] Weihua Hu*, Bowen Liu*, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2020. Strategies for Pre-training Graph Neural Networks. In *ICLR*.

[19] Williams Ronald J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8 (1992), 229–256.

[20] Jongmin Kim, Taesup Kim, Sungwoong Kim, and Chang D. Yoo. 2019. Edge-Labeling Graph Neural Network for Few-Shot Learning. In *CVPR*.

[21] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *ICLR* (2017).

[22] Boris Knyazev, Graham W Taylor, and Mohamed Amer. 2019. Understanding Attention and Generalization in Graph Neural Networks. In *NeurIPS*.

[23] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. MeLU: Meta-Learned User Preference Estimator for Cold-Start Recommendation. In *KDD* (Anchorage, AK, USA) *(KDD âĂŹ19)*. Association for Computing Machinery, New York, NY, USA, 1073âĂŞ1082. https://doi.org/10.1145/3292500.3330859

[24] Jaekoo Lee, Hyunjae Kim, Jongsun Lee, and Sungroh Yoon. 2017. Transfer Learning for Deep Learning on Graph-Structured Data. In *AAAI*.

[25] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-Attention Graph Pooling. In *ICML*. 3734–3743.

[26] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. 2018. Meta-SGD: Learning to Learn Quickly for Few Shot Learning. In *ICML*.

[27] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2019. Prototype Propagation Networks (PPN) for Weakly-supervised Few-shot Learning on Category Graph. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

[28] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Learning to Propagate for Graph Meta-Learning. In *NeurIPS*.

[29] LU LIU, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Learning to Propagate for Graph Meta-Learning. In *NeurIPS*. Curran Associates, Inc., 1039–1050.

[30] Alfred MÃijller. 1997. Integral Probability Metrics and Their Generating Classes of Functions. *Advances in Applied Probability* 29, 2 (1997), 429–443.

[31] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On First-Order Meta-Learning Algorithms. arXiv:cs.LG/1803.02999

[32] Feiyang Pan, Shuokai Li, Xiang Ao, Pingzhong Tang, and Qing He. 2019. Warm Up Cold-Start Advertisements: Improving CTR Predictions via Learning to Learn ID Embeddings. In *SIGIR* (Paris, France) *(SIGIRâĂŹ19)*. Association for Computing Machinery, New York, NY, USA, 695âĂŞ704. https://doi.org/10.1145/3331184.3331268

[33] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. 2020. Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. In *ICLR*.

[34] Sachin Ravi and Hugo Larochelle. 2017. Optimization as a Model for Few-Shot Learning. In *ICLR*.

[35] Kaspar Riesen and Horst Bunke. 2008. "IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning". In *Structural, Syntactic, and Statistical Pattern Recognition*, Niels da Vitoria Lobo, Takis Kasparis, Fabio Roli, James T. Kwok, Michael Georgiopoulos, Georgios C. Anagnostopoulos, and Marco Loog (Eds.).

[36] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. 2019. Meta-Learning with Latent Embedding Optimization. In *ICLR*.

[37] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-Learning with Memory-Augmented Neural Networks. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 1842–1850. http://proceedings.mlr.press/v48/santoro16.html

[38] Victor Garcia Satorras and Joan Bruna Estrach. 2018. Few-Shot Learning with Graph Neural Networks. In *ICLR*.

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

[40] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, David van Dyk and Max Welling (Eds.), Vol. 5. PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 488–495.

[41] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical Networks for Few-shot Learning. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4077–4087. http://papers.nips.cc/paper/6996-prototypical-networks-for-few-shot-learning.pdf

[42] B. K. Sriperumbudur, K. Fukumizu, A. Gretton, B. SchÃűlkopf, and G. R. G. Lanckriet. 2010. Non-parametric estimation of integral probability metrics. In *2010 IEEE International Symposium on Information Theory*. 1428–1432.

[43] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. 2017. A Meta-Learning Perspective on Cold-Start Recommendations for Items. In *NeurIPS*. Curran Associates, Inc., 6904–6914.

[44] Petar VeliÄŊkoviÄĞ, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro LiÃš, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[45] Petar VeliÄŊkoviÄĞ, William Fedus, William L. Hamilton, Pietro LiÃš, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR*.

[46] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.

[47] Ling Yang, Liangliang Li, Zilun Zhang, Xinyu Zhou, Erjin Zhou, and Yu Liu. 2020. DPGN: Distribution Propagation Graph Network for Few-shot Learning. arXiv:cs.CV/2003.14247

[48] Huaxiu Yao, Xian Wu, Zhiqiang Tao, Yaliang Li, Bolin Ding, Ruirui Li, and Zhenhui Li. 2020. Automated Relational Meta-learning. In *ICLR*.

[49] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh V. Chawla, and Zhenhui Li. 2020. Graph Few-shot Learning via Knowledge Transfer. In *AAAI*.

[50] Mingzhang Yin, George Tucker, Mingyuan Zhou, Sergey Levine, and Chelsea Finn. 2020. Meta-Learning without Memorization. In *ICLR*.

[51] Chao Zhang, Lei Zhang, and Jieping Ye. 2012. Generalization Bounds for Domain Adaptation. In *NeurIPS* (Lake Tahoe, Nevada) *(NIPSâĂŹ12)*. Curran Associates Inc., Red Hook, NY, USA, 3320âĂŞ3328.

[52] Pengchuan Zhang, Qiang Liu, Dengyong Zhou, Tao Xu, and Xiaodong He. 2018. On the Discrimination-Generalization Trade off in GANs. In *ICLR*.

[53] Ruixiang ZHANG, Tong Che, Zoubin Ghahramani, Yoshua Bengio, and Yangqiu Song. 2018. MetaGAN: An Adversarial Approach to Few-Shot Learning. In *NeurIPS*. Curran Associates, Inc., 2365–2374.

[54] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. 2019. Hierarchical Graph Pooling with Structure Learning. *arXiv preprint arXiv:1911.05954* (2019).

[55] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2018. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202* (2018).

[56] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. 2019. Meta-GNN: On Few-Shot Node Classification in Graph Meta-Learning. In *CIKM* (Beijing, China) *(CIKM âĂŹ19)*. Association for Computing Machinery, New York, NY, USA, 2357âĂŞ2360.

[57] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

[58] Luisa Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. 2019. Fast Context Adaptation via Meta-Learning. In *ICML*.