Relative and Absolute Location Embedding for Few-Shot Node Classification on Graph (Technical Appendices)

Zemin Liu,[1] Yuan Fang,[1] Chenghao Liu,[1,2] Steven C.H. Hoi[1,2]

[1] Singapore Management University, Singapore;
[2] Salesforce Research Asia, Singapore
{zmliu, yfang}@smu.edu.sg, {chenghao.liu, shoi}@salesforce.com

A Dataset Details

We provide further details of the datasets. For ***Amazon***, the feature vector of each node was extracted from the images of the corresponding item. The categories of items represent the node classes. For ***Email***, the 128-dimensional feature vector of each node was generated by DeepWalk. The departments of the members represent the classes. We removed the classes with less than 10 nodes. For ***Reddit***, we constructed the feature vector of each node (post) following GraphSAGE (Hamilton, Ying, and Leskovec 2017), which consists of four parts: (1) the average embedding of the words in the title of the post; (2) the average embedding of the words in the comments of the post; (3) the score of this post; (4) the number of comments on the post. All word embeddings are 300-dimensional, computed using the off-the-shelf GloVe CommonCrawl word vectors. The topical community of each post serves as the class.

B Algorithms and Complexity Analysis

We summarize the overall training of RALE in Alg. 1, and the subroutine of loss calculation for a given node set in Alg. 2. Subsequently, we analyze the complexity.

B.1 Algorithms

Alg. 1 contains the overall training process of RALE. For a task t, we first adapt the prior Θ on support set S_t and obtain Θ' in lines 6–7; then we calculate the task loss on the query set Q_t based on Θ' in line 8 and accumulate to the loss of the batch in line 9. The batch-level loss is backpropagated to update the prior in line 11.

In Alg. 2, we calculate the loss on a target node set O_t. Here O_t could be either S_t or Q_t, in order to calculate the support or query loss, respectively. Specifically, in lines 2–3, for each node $v \in O_t$ we first gather the connecting paths $\mathcal{P}_{S_t,v}$ and $\mathcal{P}_{\mathcal{H},v}$ for the task-level RL embedding and graph-level AL embedding, respectively. Next, we obtain the node embedding and the RL and AL embedding of v in lines 4–6, which are concatenated in the dependency-aware classification layer in line 7. Finally, in line 10, the cross-entropy loss for all nodes in the target set O_t is computed given the classification layer.

Algorithm 1 MODEL TRAINING FOR RALE

```
Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathcal{C}, \ell)$, base classes $\mathcal{C}_b$, hub set $\mathcal{H}$, task
    adaptation learning rate $\alpha$, meta-learning rate $\beta$.
Output: Prior $\Theta$.
 1: $\Theta \leftarrow$ parameter initialization;
 2: while not converged do
 3:     sample a batch of tasks $\mathcal{T}$ w.r.t. $\mathcal{G}$ and $\mathcal{C}_b$;
 4:     $L_\mathcal{T} \leftarrow 0$;                                ▷ loss of the batch
 5:     for each task $t \in \mathcal{T}$ do
 6:         $L(S_t; \Theta) \leftarrow$ LOSSCALCULATION($\mathcal{G}, t, S_t, \Theta, \mathcal{H}$);
 7:         $\Theta' = \Theta - \alpha \frac{\partial L(S_t;\Theta)}{\partial \Theta}$;                ▷ Eq. (10)
 8:         $L(Q_t; \Theta') \leftarrow$ LOSSCALCULATION($\mathcal{G}, t, Q_t, \Theta', \mathcal{H}$);
 9:         $L_\mathcal{T} \leftarrow L_\mathcal{T} + L(Q_t; \Theta')$;
10:     end for
11:     $\Theta \leftarrow \Theta - \beta \frac{\partial L_\mathcal{T}}{\partial \Theta}$;                ▷ optimizing Eq. (11)
12: end while
13: return $\Theta$.
```

Algorithm 2 LOSSCALCULATION

```
Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathcal{C}, \ell)$, task $t$, target node set $O_t$ (i.e.,
    either $S_t$ or $Q_t$), parameters $\Theta$, hub set $\mathcal{H}$.
Output: $L(O_t; \Theta)$.
 1: for $v \in O_t$ do
 2:     $\mathcal{P}_{S_t,v} \leftarrow \{\mathcal{P}_{s,v} : s \in S_t\}$;                ▷ paths for RL
 3:     $\mathcal{P}_{\mathcal{H},v} \leftarrow \{\mathcal{P}_{h,v} : h \in \mathcal{H}\}$;                ▷ paths for AL
 4:     $\mathbf{h}_v \leftarrow \phi_g(v; \theta_g)$;                ▷ node embedding
 5:     $\mathbf{e}_v^{S_t} \leftarrow \phi(\mathcal{P}_{S_t,v}; \theta_g, \theta_p)$;                ▷ RL embedding, Eq. (6)
 6:     $\mathbf{e}_v^{\mathcal{H}} \leftarrow \phi(\mathcal{P}_{\mathcal{H},v}; \theta_g, \theta_p)$;                ▷ AL embedding, Eq. (7)
 7:     $\psi(v; \Theta) \leftarrow$ SOFTMAX $\left(\sigma(\mathbf{W}\left[\mathbf{h}_v \| \mathbf{e}_v^{S_t} \| \mathbf{e}_v^{\mathcal{H}}\right])\right)$;
 8:                                ▷ classification layer, Eq. (8)
 9: end for
10: $L(O_t; \Theta) \leftarrow -\sum_{v \in O_t} \sum_{i=1}^{m} \mathbb{I}_{\ell(v)=i} \ln(\psi(v; \Theta)[i])$;
11:                                ▷ cross-entropy loss, Eq. (9)
12: return $L(O_t; \Theta)$.
```

B.2 Complexity Analysis

The training algorithm contains two stages, as discussed in the main paper. In what follows, we analyze the complexity of each stage.

First stage: offline construction of paths. This stage involves the sampling of path segments and the construction of paths, which only needs to be done once as precomputation. In particular, during the computation of the support or query loss, the paths gathered for RL and AL embedding (lines 2–3 in Alg. 2) are precomputed in this stage. We analyze these two sub-parts separately.

1) Segment sampling: We first sample w paths with length l for each node with complexity $O(|\mathcal{V}| \cdot w \cdot l)$; for each sampled path, we further apply a sliding window of size $l_p/2$ to extract path segments up to length $l_p/2$. Thus, the overall complexity of segments sampling is $O(|\mathcal{V}| \cdot w \cdot l \cdot l_p)$.

2) Path construction: Given a pair of nodes, we join their sampled segments that end with a common hub into full paths up to length l_p. As we store the end node of each segment as the key and the segment as the value, the retrieve of the segments can be done in $O(1)$ time. Thus, to construct up to n_p paths between each pair of nodes, the complexity is $O(n_p)$, where n_p is set to a constant 10 in our experiments (*i.e.*, for each pair, we only consider at most $n_p = 10$ paths between them).

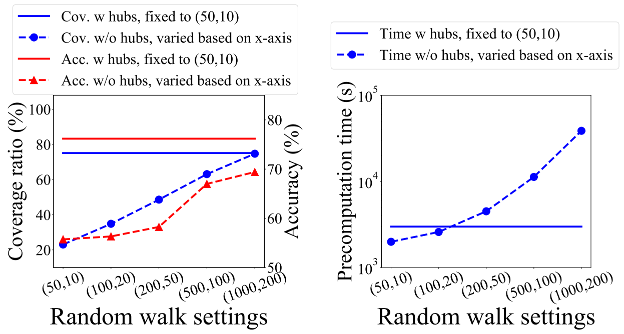
Second stage: online model training. The main component of training involves the calculation of the concatenated representation in the classification layer, *i.e.*, $[\mathbf{h}_v \| \mathbf{e}_v^{S_t} \| \mathbf{e}_v^{\mathcal{H}}]$ in Eq. (8). We analyze its complexity for a target node in an m-way k-shot task. Given average node degree d and number of aggregation layers n_l in the graph encoder (*i.e.*, GNN), the embedding of the target node $\mathbf{h}_v$ can be calculated in $O(d^{n_l})$ time. Moreover, to calculate the RL and AL embedding $\mathbf{e}_v^{S_t}$ and $\mathbf{e}_v^{\mathcal{H}}$, we need to first analyze the complexity of the path encoder between the target node and a reference node. Given n_p paths each with a length up to l_p, the complexity of the path encoder is $O(n_p \cdot l_p \cdot d^{n_l})$. Note that there is a factor of d^{n_l}, since the path encoder requires the output from the graph encoder for every node in the paths. While RL embedding takes the set of support nodes S_t as references, AL embedding takes the set of hubs $\mathcal{H}$ as references. Thus, the complexity of RL and AL embedding is $O((|S_t|+|\mathcal{H}|) \cdot n_p \cdot l_p \cdot d^{n_l})$. More specifically, for RL embedding $|S_t| = mk$ in an m-way k-shot task, and for AL embedding we only sample $|\tilde{\mathcal{H}}|$ hubs for efficiency, where $|\tilde{\mathcal{H}}|$ is set to a constant 10 in our experiments. Thus, the overall complexity of calculating the concatenated representation is $O((mk + |\tilde{\mathcal{H}}|) \cdot n_p \cdot l_p \cdot d^{n_l})$. Note that for Meta-GNN (Zhou et al. 2019), the complexity of computing the embedding for a target node is $O(d^{n_l})$.

C Additional Experiments

We include additional experimental results on the further analysis of hubs, as well as the training time.

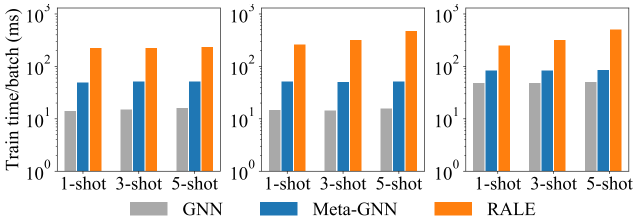
C.1 Further Analysis of Hubs

As discussed in the main paper, hubs facilitate the sampling of paths. To sample between a pair of nodes (u, v), without hubs, the vast majority of random walks starting u will not reach v, resulting in very low sampling efficiency. In this experiment, we compare RALE with an alternative strategy without using hubs. For RALE, we fix its random walk setting as (50,10) (*i.e.*, sample $w = 50$ random walks of length $l = 10$ starting from each node). In the alternative strategy, in order to obtain a reasonable number of samples with a similar coverage ratio to RALE, more and longer random walks must be performed. We report the findings in Fig. I, under the 1-shot setting on the Amazon dataset. In particular, we compare the paths coverage ratio and accuracy in Fig. I(a) and the path precomputation time in Fig. I(b).



(a) Coverage and accuracy. (b) Precomputation time.

Figure I: Analysis of random walk settings.



(a) Amazon (b) Email (c) Reddit

Figure II: Comparison of training time per batch.

Several conclusions can be drawn. Firstly, using the same random walk setting (50, 10) as RALE, without hubs the coverage ratio and accuracy are significantly lower than RALE. Secondly, we increase the random walk setting from (50, 10) to (1000, 200) for the strategy without hubs, in order to sample more paths. As expected, the coverage ratio and accuracy both increase. While at (1000, 200) the strategy without hubs can ultimately achieve comparable coverage ratio with RALE at only (50, 10), its accuracy still falls short. A potential reason is that, although they achieve similar coverage ratio, hub-passing paths typically have elevated significance due to the structural importance of hubs, and hubs are also crucial to the graph-level AL embedding. Thirdly, although without hubs we can eventually improve the coverage ratio, the corresponding precomputation time increases drastically. At (1000, 200) the strategy without hubs incurs a more than 10-fold increase in precomputation time, as compared to RALE at (50, 10). Overall, the results demonstrate the effectiveness and efficiency of using hubs.

C.2 Training Time Comparison

We compare the training time for a batch of tasks among the bare GNN (*i.e.*, without task adaptation), Meta-GNN and RALE. As shown in Fig. II, we present the average training time per batch.

We make the following observations. Firstly, GNN has the lowest training cost, since it does not need any adaptation on the support set in each task. With adaptation on the support set, Meta-GNN and RALE both cost more than GNN. Secondly, RALE incurs more training time than Meta-GNN, which is attributed to the path encoding for RL and AL embedding. Thirdly, as the number of shots increases, the time cost generally increases, especially for RALE since there are more pairs of nodes for path encoding and path encoding dominates the cost.

D Experimental Environment

We use Python-3.6.5 and Tensorflow-1.13.1 for implementation, and run all the experiments on a single workstation with a Xeon W-2133 CPU, an RTX 2080Ti GPU and 128GB memory.