Towards Locality-Aware Meta-Learning of Tail Node Embeddings on Networks

Zemin Liu[1*], Wentao Zhang[2*†], Yuan Fang[1], Xinming Zhang[2], Steven C.H. Hoi[1,3‡]
[1]Singapore Management University, [2]University of Science and Technology of China, [3]Salesforce Research Asia
{zmliu, yfang}@smu.edu.sg, zwt@mail.ustc.edu.cn, xinming@ustc.edu.cn, shoi@salesforce.com

ABSTRACT

Network embedding is an active research area due to the prevalence of network-structured data. While the state of the art often learns high-quality embedding vectors for high-degree nodes with abundant structural connectivity, the quality of the embedding vectors for low-degree or *tail* nodes is often suboptimal due to their limited structural connectivity. While many real-world networks are long-tailed, to date little effort has been devoted to tail node embedding. In this paper, we formulate the goal of learning tail node embeddings as a *few-shot regression* problem, given the few links on each tail node. In particular, since each node resides in its own local context, we personalize the regression model for each tail node. To reduce overfitting in the personalization, we propose a *locality-aware* meta-learning framework, called *meta-tail2vec*, which learns to learn the regression model for the tail nodes at different localities. Finally, we conduct extensive experiments and demonstrate the promising results of meta-tail2vec. (Supplemental materials including code and data are available at https://github.com/smufang/meta-tail2vec.)

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Computing methodologies** → **Machine learning**; **Learning latent representations**.

KEYWORDS

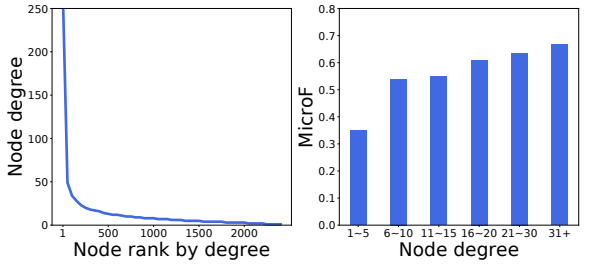
meta-learning, network embedding, tail nodes

ACM Reference Format:
Zemin Liu, Wentao Zhang, Yuan Fang, Xinming Zhang, Steven C.H. Hoi. 2020. Towards Locality-Aware Meta-Learning of Tail Node Embeddings on Networks. In *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/10.1145/3340531.3411910

*Co-first authors with equal contribution.
†Work done as a visiting research student at Singapore Management University.
‡Currently on leave from Singapore Management University.

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

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411910



(a) Degree distribution (b) Classification performance

Figure 1: Distribution of node degree and its relationship to the quality of embedding vector on the Wiki network.

1 INTRODUCTION

Network structures are prevalent in various real-world scenarios, such as social networks, citation networks and biological networks. On these networks, many problems can be formulated as node classification and link prediction, which rely heavily on effective network representations. While traditional approaches mainly focus on manual feature engineering, recent network embedding [26] and graph neural networks [17] have become the *de facto* state-of-the-art approaches for learning representations of network data. Specifically, these methods aim to encode network structures by mapping nodes into a low-dimensional vector space, in which the structural information is preserved.

Despite their success, a critical question remains open: the performance of most existing embedding methods depends on the availability of abundant structural connectivity. Although this is not a concern for medium-to-high degree nodes with many links to other nodes, low-degree nodes with very few links often suffer from the problem of limited structural information. As a result, the embedding vector for a low-degree node cannot be accurately learned from its structural information. Generally, the node degrees vary considerably across the network and are not uniformly distributed. In many networks, the degrees approximately follow the power-law distribution [26]. For instance, Fig. 1(a) illustrates the degree distribution of the Wiki network [39], a network of interlinked Wikipedia pages belonging to 19 categories. The node degrees are characterized by a long-tailed distribution, where a significant fraction of the nodes belong to the tail with very low degrees. The embeddings of these *tail* nodes are unsatisfactory, as demonstrated by the performance of node classification in Fig. 1(b). Particularly, when the node degree decreases, the performance also decreases due to less structural information.

Unfortunately, most current network embedding or graph neural network approaches overlook the tail nodes, by regarding all

nodes uniformly and adopting the same learning approach despite their diverse degrees. For example, DeepWalk [26] samples node sequences from the network and feeds them to the same skip-gram model without paying special attention to low-degree nodes, while graph neural networks [12, 17] derive the embedding of a target node by aggregating its neighboring nodes in the same manner without adapting to the degree of the target node. Not surprisingly, the tail nodes with very few links are usually under-modeled than those with many links. This inspires us to investigate the following research problem: *How do we learn effective embedding vectors for tail nodes from limited structural information?*

The problem is challenging for two reasons. (1) The tail nodes have very few links, which provide *scarce structural information*. The issue is especially severe when only network structures are available, without access to additional side information such as item contents in recommendation [36] and morphological information in word embedding [1]. (2) Each node is associated with a unique *locality*, and thus the embedding model should ideally become specialized from node to node. However, in practice this presents a dilemma: adapting to each node often causes overfitting, which is particularly serious for tail nodes with very few observed links.

To address the first challenge of scarce structural information, we exploit the embeddings of nodes with a sufficient number of links (*i.e.*, non-tail nodes and we call them *head* nodes hereafter). Considering any network embedding model, the learned embedding vectors of head nodes tend to be more accurate than those of tail nodes, due to more abundant structural information associated with head nodes. Thus, we treat the embeddings of head nodes as our *oracle embeddings*, which can be leveraged to train a regression model capable of *reconstructing* the oracle embeddings in a self-supervised manner. Using the regression model, our goal is to predict new embeddings for tail nodes such that their quality can approach that of the oracle embeddings. To further ensure that the regression model fits both head and tail nodes, we propose to perform *link dropouts* on head nodes, to simulate the limited structural information of tail nodes.

To address the second challenge of adapting to the locality of each node, we resort to the meta-learning paradigm [9, 28], for its ability of adapting to new learning tasks (also called episodes) by learning a transferable prior common to different tasks. Known as *meta-learning*, it learns how to learn a model in the form of a prior, contrary to learning one model for all tasks (which cannot differentiate tasks) or individual models for each task (which may overfit to each task). In particular, we adopt the meta-learning framework MAML [9], which has the advantage of quick adaptation in a model-agnostic manner and has achieved considerable success in various problem domains [15, 19].

To realize the above insights in our scenario, we construct *locality-aware* tasks: one task for each node to represent its locality, and learn a regression prior that can be easily adapted to each task or locality. In each task, given a *query* node, we aim to predict its embedding vector after training on a set of *support* nodes. While the support set is often randomly sampled, for locality-aware learning, we propose to use the neighbors of the query node as the support nodes, assuming that the localities of neighboring nodes are similar. More specifically, during meta-training, each task involves a head node as the query, with the objective of learning a regression prior such that the prior can be adapted by the support nodes to accurately reconstruct the oracle embeddings of the query. During meta-testing, each task involves a tail node as the query, and the learned prior will be adapted by the support nodes before predicting a better embedding vector for the query. To make the meta-training and meta-testing tasks more similar, we again adopt link dropouts to sample only a few neighbors as the support set during meta-training, to simulate meta-testing tasks on tail nodes. Essentially, each task is a *few-shot* regression problem.

In summary, we propose a novel approach *meta-tail2vec*, which learns to learn tail node embeddings in a few-shot regression setting. The model operates in an embedding-agnostic manner, which is flexible to work with any network embedding model. Specifically, our contribution is threefold. (1) We formulate the novel problem of learning tail node embeddings on networks, and cast it as an instance of regression via oracle reconstruction. (2) We propose a base regression model hinged on the concept of link dropouts, and formulate the locality-aware tasks on networks in a meta-learning framework, which allows for easy local adaption of the base model. (3) We conduct extensive experiments on three public datasets, in which meta-tail2vec attains significant performance gains.

2 RELATED WORK

Network embedding [2] has been popular for graph representation learning, requiring only network structures as input. Various approaches have been proposed to generate structure-preserving node embeddings, such as random walks [11, 26], matrix factorization [4, 23], node proximity [31], and motifs or subgraphs [7, 38]. Graph neural networks (GNN) [12, 17] emerge as another powerful tool for representation learning, which are designed to integrate both node features and network structures.

However, little attention has been paid to the learning of tail node embeddings. While several recent studies explore various techniques to deal with sparse networks, such as using dual dropouts to avoid overfitting [3], adopting the adversarial principle to learn the underlying distribution [25, 35] and leveraging the global network structure [34], they aim to increase the overall robustness and do not specifically target the most vulnerable tail nodes. Nevertheless, some studies also deal with sparse observations in other domains, such as cold-start recommendation [20, 33, 36] and rare word embedding [1, 15, 27]. However, these methods rely on additional side information, such as item contents or word morphology.

To address the general problem of learning from less data, particularly in few-shot scenarios, meta-learning [9, 29, 32] has demonstrated considerable success in several domains including vision [32], language [13, 15] and data mining [19, 24]. These methods typically learn some prior knowledge from an abundant number of related tasks, and adapt the prior to new tasks with limited data. As many forms of data often entail inherent graph structures, recent studies often exploit some auxiliary graphs, such as affinity graphs [10, 22], class relational graphs [21] and one-hop ego-networks [37, 42]. Few-shot learning on graphs proper have also been explored, including node classification on a single large graph [43], and node or graph-level classification on multiple graphs [5, 40]. To the best of our knowledge, our approach is the first use of meta-learning for tail node embedding.

3 PROBLEM CASTING AS REGRESSION

We begin with the problem statement, and cast it as an instance of regression based on the idea of oracle reconstruction.

3.1 Problem statement

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the set of nodes $\mathcal{V}$ and the set of edges $\mathcal{E}$. Let $\mathcal{N}_v$ denote the set of neighbors of node $v \in \mathcal{V}$, and the size of the neighbor set $|\mathcal{N}_v|$ is known as the degree of v. In particular, v is a low-degree or *tail* node if its degree is no larger than some constant k. That is, the set of tail nodes is $\mathcal{V}_{\text{tail}} = \{v \in \mathcal{V} : |\mathcal{N}_v| \leq k\}$. We call the remaining nodes *head* nodes: $\mathcal{V}_{\text{head}} = \{v \in \mathcal{V} : |\mathcal{N}_v| > k\}$.

Given any base network embedding model ϕ, we denote $\mathbf{h}_v = \phi(\mathcal{G}, v) \in \mathbb{R}^d$ as the d-dimensional embedding vector of node v, learned by ϕ on graph $\mathcal{G}$, $\forall v \in \mathcal{V}$. Since the tail nodes have scarce structural features due to their small degrees, presumably their embeddings $\{\mathbf{h}_v : v \in \mathcal{V}_{\text{tail}}\}$ are inferior in quality to the embeddings of the head nodes $O = \{\mathbf{h}_v : v \in \mathcal{V}_{\text{head}}\}$.

Our goal is to learn new embedding vectors for the tail nodes $\{\hat{\mathbf{h}}_v : v \in \mathcal{V}_{\text{tail}}\}$, such that their quality is improved to eventually approach the quality of the head nodes' embeddings O. Note that our setup is embedding-agnostic, *i.e.*, any embedding model ϕ can be used as the base embedding model.

3.2 Regression via oracle reconstruction

A major challenge of learning tail node embeddings lies in the limited availability of structural information. That is, each tail node only has a few observed neighbors. Without assuming additional side information (*e.g.*, item contents in recommendation systems), we take advantage of the high-quality embeddings of head nodes, and investigate how high-quality embeddings can be similarly constructed for the tail nodes.

Specifically, we propose to cast the problem as an instance of regression. We train a regression model F on the head nodes, treating their embeddings O, which are learned by the base embedding model ϕ, as the *oracle embeddings* w.r.t. ϕ. Given any head node v, $F(v; \Theta)$ outputs a new predicted embedding $\hat{\mathbf{h}}_v$ to approximate its oracle embedding $\mathbf{h}_v \in O$. In other words, F is expected to *reconstruct* the oracle embeddings O of the head nodes, regardless of how the base embedding model works. Formally, we optimize the parameters Θ of the regression model F by

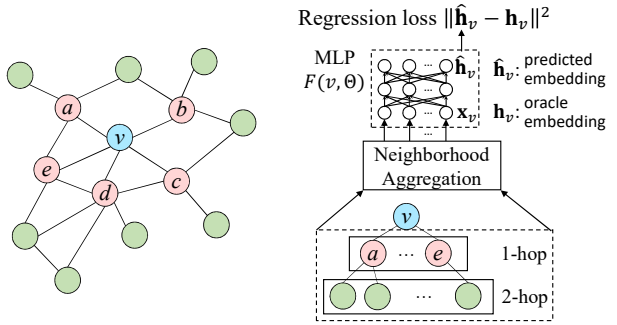
$$\arg\min_{\Theta} \sum_{v \in \mathcal{V}_{\text{head}}} \|F(v; \Theta) - \mathbf{h}_v\|^2. \tag{1}$$

Note that here we use the Euclidean norm $\|\cdot\|$, although other distance functions can be adopted too.

After training, F can be applied to the tail nodes as test instances, so as to predict their unknown oracle embeddings, supposedly attaining similar quality as O. The improved embedding vectors of tail nodes can be subsequently used as representations for downstream tasks such as node classification and link prediction, whose performance would benefit from the better embeddings.

4 META-LEARNED FEW-SHOT REGRESSION

In this section, we first introduce a regression model to predict new embedding vectors for tail nodes, then personalize the regression for each node by formulating node-wise locality-aware tasks, and finally present the overall meta-learning approach *meta-tail2vec*.



(a) 2-hop neighbors of node v (b) Embedding regression model

Figure 2: Training a regression model to predict node embeddings reconstructed from neighboring nodes.

4.1 Embedding regression model

As illustrated in Fig. 2, we materialize the regression model $F(v; \Theta)$ in order to reconstruct the oracle embedding $\mathbf{h}_v$ of a node v. Specifically, we leverage the widely used Multi-Layer Perceptron (MLP):

$$\hat{\mathbf{h}}_v = F(v; \Theta) = W_2 \cdot \sigma(W_1 \mathbf{x}_v + \mathbf{b}_1) + \mathbf{b}_2, \tag{2}$$

where $\mathbf{x}_v \in \mathbb{R}^{d_1}$ is the input feature vector of node v. The parameters of the MLP include $W_1 \in \mathbb{R}^{d_2 \times d_1}$, $W_2 \in \mathbb{R}^{d \times d_2}$, $\mathbf{b}_1 \in \mathbb{R}^{d_2}$ and $\mathbf{b}_2 \in \mathbb{R}^d$, *i.e.*, $\Theta = \{W_1, W_2, \mathbf{b}_1, \mathbf{b}_2\}$. $\sigma(\cdot)$ is an activation function, and we adopt ReLU in this paper. Note that the size of the output layer is d, the same as the embedding dimension of the nodes.

For a node v, the input to the MLP is a feature vector $\mathbf{x}_v$. In the following, we discuss the formulation of the input feature vector.

4.1.1 Neighborhood aggregation. On a network, a node v is characterized by its structural features or its neighbor set, $\mathcal{N}_v$. Naturally, we can aggregate the embedding vectors of the neighbors to construct its input feature $\mathbf{x}_v$, an idea similar and central to many graph neural networks [12, 17]. Specifically,

$$\mathbf{x}_v = \textsc{Aggr}(\{\mathbf{h}_i : i \in \mathcal{N}_v\}), \tag{3}$$

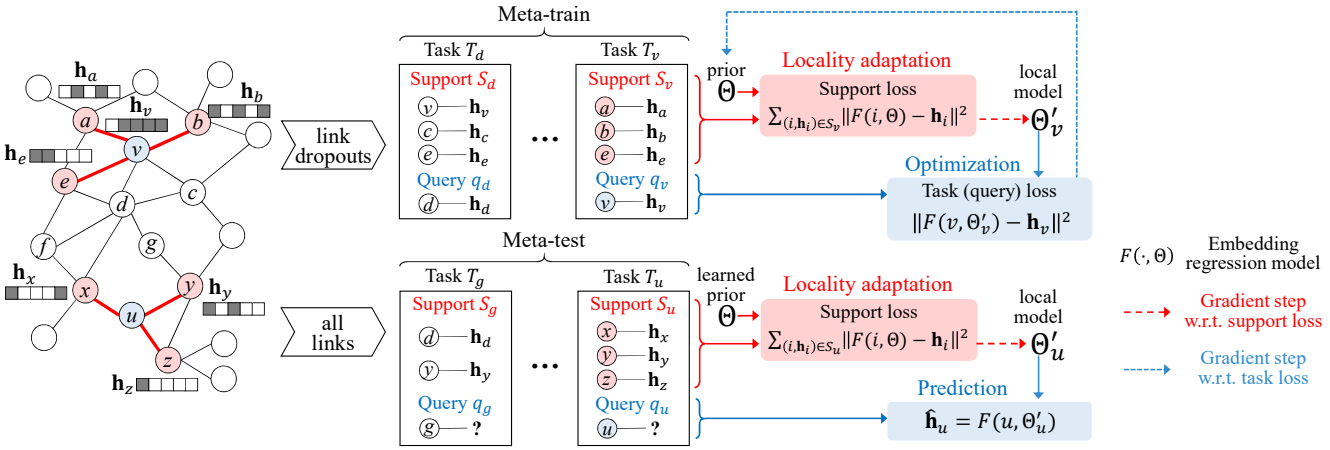
where $\textsc{Aggr}(\cdot)$ is an aggregator such as mean pooling. More generally, we can also aggregate nodes within an m-hop radius of v, as shown in Fig. 2(a). Defining the m-hop neighbor set of v as

$$\mathcal{N}_v^{(m)} = \bigcup_{i \in \mathcal{N}_v^{(m-1)}} \mathcal{N}_i, \tag{4}$$

and $\mathcal{N}_v^{(1)} \equiv \mathcal{N}_v$, the input feature $\mathbf{x}_v$ can be constructed as

$$\mathbf{x}_v = \textsc{Aggr}(\{\mathbf{h}_i : i \in \mathcal{N}_v^{(1)} \cup \mathcal{N}_v^{(2)} \cup \ldots \cup \mathcal{N}_v^{(m)}\}). \tag{5}$$

Note that, although advanced strategies such as tree-LSTMs [30] and graph convolutions [17] may be employed to aggregate multiple hops, we adopt the simple mean pooling to demonstrate the effectiveness of our approach.



(a) Toy network (b) Locality-aware tasks (c) Locality-aware meta-learning

Figure 3: Overall framework of our locality-aware tail node embedding model meta-tail2vec. (Best viewed in color.)

4.1.2 Link dropouts. One major flaw of such an input vector $\mathbf{x}_v$ is that the head nodes in training and the tail nodes in testing posses very different neighbor sets in terms of their abundance, *i.e.*, $|N_v| \gg |N_u|$ for some $v \in \mathcal{V}_{\text{head}}$ and $u \in \mathcal{V}_{\text{tail}}$. To make the training and testing nodes more similar, we perform *link dropouts* on the head nodes. Specifically, we randomly sample only k neighbors of each head node for aggregation, in order to simulate the tail nodes. That is, given the sampled neighbors $\widetilde{\mathcal{N}}_v$ of a head node v, for one-hop aggregation we have

$$\mathbf{x}_v = \textsc{Aggr}(\{\mathbf{h}_i : i \in \widetilde{\mathcal{N}}_v\}), \quad \forall v \in \mathcal{V}_{\text{head}}. \tag{6}$$

The idea of link dropouts draws an interesting parallel to DropoutNet [33]. In their approach designed for cold-start recommendation, they eliminate the structural factor of users or items so as to simulate a cold-start scenario, forcing the model to fall back to user or item contents. However, we do not assume any side information like user contents, which means our problem is more challenging and we can only drop the structural information partially.

4.2 Locality-aware few-shot regression tasks

When applying the regression model on all nodes, it ignores the unique locality of each node. As the nodes reside across different localities on the network, assuming one global model to fit all nodes is unrealistic. At the other extreme, learning an individual model for each node, including the popular pre-training and fine-tuning strategy [6], is likely to cause severe overfitting due to the limited structural information at the locality of each tail node.

4.2.1 Locality-aware support sets. To address the challenge of adapting to the locality of each node, we resort to the episodic meta-learning paradigm [9, 28]. The framework consists of many similar-natured learning tasks, divided into *meta-training* and *meta-testing* tasks. While each task is an instance in the meta-learning process, each task itself is a learning problem consisting of *support* and *query* sets (representing the usual sense of training and testing data, respectively). The goal of the meta-learning is to extract prior knowledge common to all tasks in meta-training, such that the knowledge can be quickly adapted to new tasks in meta-testing. In other words, it learns how to learn a task in the form of a common prior, instead of directly learning each task.

In our context, each task represents the unique locality of a node. At the task level, given a *query* node, we aim to predict its embedding vector after training on a set of *support* nodes. At the meta-learning level, we learn a prior regression model F parameterized by Θ from the meta-training tasks where the query of each task is a head node, and further adapt the prior to the learning of new tasks in meta-testing where the query of each task is a tail node. However, in traditional meta-learning tasks [9, 43], the support set for a query is randomly sampled. In such random sampling, the support set is not related to the query, and thus cannot reflect the unique locality of each query node. To generate locality-aware tasks, we propose to use the neighbors of the query node as the support nodes. The assumption is that the localities of neighboring nodes are similar, and thus training on these support nodes would be also applicable to the query node which lies in the vicinity of the support nodes. In other words, in each of our tasks, the support and query nodes are coupled based on their locality.

4.2.2 Formulation of few-shot tasks. As the neighbor sets of head and tail nodes differ vastly in size, a meta-training task with a head node as the query has many support nodes, and a meta-testing task with a tail node as the query has few support nodes. To make the tasks more similar, we again adopt link dropouts by sampling only k neighbors as the support set for meta-training tasks to simulate meta-testing tasks. Thus, each task becomes a *few-shot* (up to k shots) regression problem, to predict the embedding vector of a query node from a few (up to k) support nodes.

We illustrate the task formulation with an example. As shown in Fig. 3(a), assuming $k = 3$, v and u is a head and tail node, respectively. On the one hand, the head node v will be used to formulate a meta-training task: v itself will be the query node, whereas we sample k nodes from v's neighbors $\mathcal{N}_v = \{a, b, c, d, e\}$, to form the support set $\widetilde{\mathcal{N}}_v$, say, $|\widetilde{\mathcal{N}}_v| = \{a, b, e\}$. On the other hand, the tail node u will be used to formulate a meta-testing task: u itself will be the query node, whereas we simply take all of u's neighbors $\mathcal{N}_u = \{x, y, z\}$ as the support set. More example tasks are illustrated in Fig. 3(b).

Formally, for each head node v, we define a meta-training task $T_v = (S_v, q_v)$ where $S_v = \{(i, \mathbf{h}_i) : i \in \widetilde{\mathcal{N}}_v\}$ is the support set, and $q_v = (v, \mathbf{h}_v)$ is the query. For each tail node u, we define a meta-testing task $T_u = (S_u, q_u)$ where $S_u = \{(i, \mathbf{h}_i) : i \in \mathcal{N}_u\}$ is the support set, and $q_u = (u, ?)$ is the query, where the embedding vector of u is unknown and to be predicted. Thus, the set of all meta-training tasks is $\mathcal{T}_{\text{train}} = \{(S_v, q_v) : v \in \mathcal{V}_{\text{head}}\}$, and the set of all meta-testing tasks is $\mathcal{T}_{\text{test}} = \{(S_u, q_u) : u \in \mathcal{V}_{\text{tail}}\}$,

In practice, we further ensure that all nodes in the support sets are head nodes, so that they all associate with an oracle embedding for adaptation to the regression model. Specifically, in meta-training, we only sample head nodes from the neighbors as the support; in meta-testing, we filter tail nodes from the support. In the rare case that all of a tail node's neighbors are also tail nodes, we will not attempt to improve its original embedding.

4.3 Meta-learning of tail node embeddings

We employ MAML [9] for the meta-learning of tail node embeddings, which is capable of learning a prior Θ for any model using gradient-based optimization. In our case, the prior is the embedding regression model F, parameterized by Θ. Different from the simple pre-training of a model, the prior Θ is learned in such a way that Θ can be quickly adapted to a new task by performing just one or a few gradient updates on the support set of the new task. The model Θ' adapted from the prior Θ, is a local model for the query node in the same task.

More specifically, in our meta-training, consider a task $T_v = (S_v, q_v)$. As show in Fig. 3(b) and (c), the prior Θ itself is not directly updated or optimized by the support set S_v. Instead, it will be adapted by S_v to produce a local model Θ'_v for the query v, through one or a few gradient updates w.r.t. S_v's loss. The adapted local model Θ'_v will be applied to the query v to predict an embedding vector $\hat{\mathbf{h}}_v$, so that the prior Θ can be updated by minimizing the task loss, *i.e.*, the distance between the predicted embedding $\hat{\mathbf{h}}_v$ and the oracle embedding $\mathbf{h}_v$ of the query node v.

Formally, let the loss of the prior on the support set S_v be

$$L_{S_v}(\Theta) = \sum_{(i,\mathbf{h}_i)\in S_v} \|F(i;\Theta) - \mathbf{h}_i)\|^2. \tag{7}$$

The prior Θ will be adapted by S_v by one (or a few) gradient updates to generate a local model Θ'_v for the task T_v. That is,

$$\Theta'_v = \Theta - \alpha \frac{\partial L_{S_v}(\Theta)}{\partial \Theta}, \tag{8}$$

where α is the learning rate for the adaptation. Afterwards, the local model Θ'_v will be applied on the query v, to calculate the task loss using the query q_v:

$$L_{q_v}(\Theta'_v) = \|F(v;\Theta'_v) - \mathbf{h}_v\|^2. \tag{9}$$

Subsequently, the prior Θ can be updated during meta-training by minimize the query-based total loss of all meta-training tasks. Given the set of meta-training tasks $\mathcal{T}_{\text{train}}$, we optimize the following to obtain the optimal prior:

$$\arg\min_{\Theta} \sum_{T_v=(S_v,q_v)\in\mathcal{T}_{\text{train}}} L_{q_v}\left(\Theta - \alpha \frac{\partial L_{S_v}(\Theta)}{\partial \Theta}\right) \tag{10}$$

Table 1: Summary of datasets.

	# nodes	# edges	# node classes	multi-label	# tail nodes
Wiki	2,405	17,981	19	No	1,069
Flickr	80,513	5,899,882	195	Yes	9,367
Email	1,005	25,571	42	No	235

On the other hand, in our meta-testing, consider a task $T_u = (S_u, q_u)$. The prior Θ, learned from meta-learning, will be adapted on the support set S_u to produce a local model Θ'_u in the same way as Eq. (8). The local model Θ'_u will be simply applied on the query u, which is a tail node, to predict a new embedding vector $\hat{\mathbf{h}}_u = F(u;\Theta'_u)$ as the output of meta-tail2vec.

Optimization and complexity. To adapt to the support set of each task in Eq. (8), we apply the standard gradient descent with one or a few steps. To minimize the meta-objective in Eq. (10), we adopt the widely used Adam optimizer. As training is performed over mini-batches of tasks, the time cost of our meta-training process depends on the total number of tasks N passed through, where each task contains up to k support nodes and the embedding of each node is aggregated from m-hop neighbors. Thus, the overall complexity is $O(Nkd^m)$, where d is the average degree of nodes. Typically m is a small constant such as 1 or 2, and k is also small by the definition of tail nodes. Furthermore, it is also common to perform neighborhood sampling [12] during the m-hop aggregation, and thus the average degree d is also effectively restricted to a constant.

5 EXPERIMENTS

In this section, we conduct node classification and link prediction on three public benchmark datasets, and evaluate the performance of our proposed meta-tail2vec.

5.1 Experimental settings

5.1.1 Datasets. We conducted experiments on three public datasets, as follows. (1) *Wiki* [39] is a network of Wikipedia pages, where each node is a page, and each edge represents the hyperlink between pages. Each page belongs to one of the 19 categories. (2) *Flickr* [26] is a network of users of the photo sharing service, where each node is a user, and each edge represents the friendship between users. Every user belongs to one or more interest groups, such as "scenic photos". (3) *Email* [41] is an e-mail network between members of a European research institution, where each node is a member, and each edge represents the communication between members. Every member belongs to one of the 42 departments. Their statistics are summarized in Table 1. Note that we regarded nodes with 5 or fewer links as the tail nodes, *i.e.*, $\{v \in \mathcal{V} : |\mathcal{N}_v| \le 5\}$.

5.1.2 Base embedding models. Our approach meta-tail2vec is flexible to work with any embedding model. We experimented with two broad categories of base embedding models. First, we employed classic network embedding and graph neural networks.

- *DeepWalk* [26]: a pioneering, widely adopted network embedding model, which samples an equal number of paths from each node, and feeds these paths into a skip-gram model to learn node embeddings.

- *GraphSAGE* [12]: a graph neural network that performs graph convolutions to aggregate features from neighboring nodes recursively. For node features, we use node embedding vectors from DeepWalk and node degrees. We adopt its self-supervised version to learn the initial base embeddings.

Second, we employed robust models designed for sparse networks.

- *SDNE* [34]: a deep network embedding model that is robust for sparse networks, by incorporating global network structures in addition to local structures.
- *ARGA* [25]: an adversarially regularized graph autoencoder, which achieves robust embedding by learning the data distribution of latent codes on the graph.
- *DDGCN* [3]: a graph convolutional network with a form of dual dropouts at both the node and edge levels, to more effectively reduce overfitting on sparse networks.

Setup and Parameters. To ensure the base models achieve their respective optimal performance, we performed a grid search to tune their parameters (optimal values in italics). For DeepWalk, we searched the number of walks $\gamma \in \{5, \textit{10}, 20\}$, walk length $t \in \{\textit{40}, 100, 150\}$ and window size $w \in \{\textit{3}, 5\}$. For GraphSAGE, we adopted a two-layer architecture, chose the aggregator from {*mean*, meanpool, maxpool} and tuned the dimension of hidden layer over {32, *64*, 128}. For SDNE, we searched the weight of local structures (as opposed to global structures) $\alpha \in \{50, \textit{100}, 150\}$ and the weight of reconstruction $\beta \in \{10, \textit{30}, 50\}$. For ARGA, we tuned the dimension of hidden layer over {16, 32, *64*}. For DDGCN, we tuned the dropout probability $p \in \{0.1, 0.3, \textit{0.5}\}$, and dual-dropout coefficient $\alpha \in \{0.1, \textit{1}, 5\}$. The optimal parameters found are generally consistent with the recommended values in the literature. For all models, the dimension of embedding vectors is set to 128.

5.1.3 Baselines for tail node refinement. We compared with a series of baselines that are also designed to improve tail node embeddings.

- *Biased walk*: Since tail nodes are under-represented, we oversampled random walks starting from the tail nodes. This method is only applicable to DeepWalk and GraphSAGE, since the other base models do not utilize random walks.
- *Additive* [18] aggregates the embeddings of the neighboring nodes as the output embedding for a tail node. We also compared to Additive-2 which aggregates the embeddings of neighboring nodes within 2 hops.
- *a la carte* [16] is a further extension of the additive model, with a transformation through an auxiliary regression task. Similarly, we also compared to a la carte-2, which aggregate the embedding of 2-hop neighbors.
- *Nonce2vec* [14] constructs a better initialization with the additive vectors, and performs another round of training using the corresponding base embedding model.
- *Dropout*: Inspired by DropoutNet [33], for each head node we sampled only k neighbors, which were further fed into an auxiliary regression task.

Setup and Parameters. The goal is to predict new embedding vectors for the tail nodes by meta-tail2vec and each baseline refinement method, w.r.t. the initial embedding vectors from each of the base embedding models.

For biased walk, we doubled the paths starting from the tail nodes compared to the head nodes. For Additive, a la carte and Nonce2vec, we used mean pooling as the aggregation function, which yields better empirical performance than min or max pooling. For a la carte and Dropout, the auxiliary regression tasks used the same regression model in our approach. For Nonce2vec, with DeepWalk, SDNE and ARGA as the base embedding model, the improved initialization was directly used as a pre-training; with GraphSAGE and DDGCN, the improved initialization was used as nodes' initial feature vectors.

For our method meta-tail2vec, we set α, the local learning rate of adapting to the support set to 0.01, and set the global meta-learning rate to 0.001. Note that it is typical to employ a larger local learning rate in order to achieve stable training [9, 43]. The number of gradient updates during adaptation was set to 5, noting that few updates such as 1 or 3 give very close results. In the regression model F, we set the size of the hidden layer of the MLP to 1024, and aggregated nodes within 2 hops. We will also analyze the impact of the number of hops in Sect. 5.3.

5.1.4 Downstream applications. We experimented with two downstream applications on the three datasets.

Node Classification. We carried out multi-class classification on Wiki and Email where each node belongs to exactly one class, and multi-label node classification on Flickr where each node can belong to one or more classes. Specifically, we evaluated the classification performance on the tail nodes, after training a logistic regression classifier on the head nodes. The predicted embeddings of the tail nodes and the oracle embeddings of the head nodes were used as their features, respectively. We adopted the evaluation metrics of micro-F and accuracy.

Link Prediction. For nodes with 2–6 links, we adopted the common leave-one-out strategy by first removing a random link from each of them (which becomes a tail node with 5 or fewer links). Our goal is to predict the removed link. On the partial network, we constructed initial embedding vectors using each base embedding model, and predicted new embedding vectors for the tail nodes with our method and each baseline refinement method. For each tail node, we treated the removed link as a positive candidate, and randomly sampled five other non-linked nodes as negative candidates. Link prediction was then formulated as a ranking problem: given a tail node, we rank its candidates using a learning-to-rank model [8] trained on the head nodes. Likewise, the predicted embeddings of the tail nodes and the oracle embeddings of the head nodes were used as their features, respectively. We adopted the evaluation metrics of mean reciprocal rank (MRR) and hit ratio at top 1 (Hit@1).

5.2 Performance comparison

We present the performance of our meta-tail2vec and various baseline refinement methods, w.r.t. each base embedding model. As the primary goal of this paper is to improve tail node embeddings, we mainly focus on comparing the performance on the tail nodes. Nevertheless, to further demonstrate that head node embeddings are not adversely impacted, we also investigate the performance on the head nodes. Note that all results are averaged over 10 runs

Table 2: Performance of node classification w.r.t. classic base embedding models.

		Base	Biased walk	Additive	Additive-2	a la carte	a la carte-2	Nonce2vec	Dropout	meta-tail2vec	Improv. over Base	Improv. over 2nd best
DeepWalk as the base embedding model												
Wiki	MicroF	44.27 ± 0.25	44.69 ± 0.31	45.32 ± 0.52	42.11 ± 0.76	23.65 ± 0.44	23.34 ± 0.47	44.97 ± 0.29	36.88 ± 0.65	**49.10** ± 0.23	+10.9%	+8.3%
	Accuracy	46.68 ± 0.31	47.05 ± 0.17	47.18 ± 0.29	44.73 ± 0.53	24.17 ± 0.49	24.48 ± 0.42	47.11 ± 0.22	38.13 ± 0.57	**50.70** ± 0.45	+8.6%	+7.5%
Flickr	MicroF	33.48 ± 0.26	33.61 ± 0.39	34.43 ± 0.41	32.59 ± 0.17	31.89 ± 0.47	32.25 ± 0.35	33.83 ± 0.28	33.91 ± 0.22	**36.31** ± 0.19	+8.5%	+5.5%
	Accuracy	32.44 ± 0.13	32.57 ± 0.19	33.29 ± 0.17	31.31 ± 0.24	32.13 ± 0.26	32.62 ± 0.31	33.01 ± 0.15	32.86 ± 0.09	**35.28** ± 0.25	+8.8%	+6.0%
Email	MicroF	51.32 ± 0.29	50.95 ± 0.24	52.50 ± 0.17	51.17 ± 0.23	17.88 ± 0.48	18.21 ± 0.52	51.84 ± 0.33	32.72 ± 0.45	**55.26** ± 0.18	+7.7%	+5.3%
	Accuracy	54.41 ± 0.34	54.13 ± 0.22	55.38 ± 0.43	53.82 ± 0.36	21.06 ± 0.45	21.13 ± 0.37	54.79 ± 0.19	33.85 ± 0.51	**57.78** ± 0.29	+6.2%	+4.3%
GraphSAGE as the base embedding model												
Wiki	MicroF	39.68 ± 0.24	40.07 ± 0.15	37.84 ± 0.31	35.96 ± 0.43	23.88 ± 0.47	22.52 ± 0.39	40.75 ± 0.33	19.78 ± 0.59	**44.29** ± 0.31	+11.6%	+8.7%
	Accuracy	41.22 ± 0.19	41.39 ± 0.06	39.31 ± 0.26	36.59 ± 0.25	25.71 ± 0.36	24.94 ± 0.62	41.65 ± 0.28	24.73 ± 0.42	**44.90** ± 0.12	+8.9%	+7.8%
Flickr	MicroF	29.38 ± 0.32	28.75 ± 0.31	27.86 ± 0.14	23.69 ± 0.44	30.02 ± 0.17	29.67 ± 0.20	29.85 ± 0.12	28.75 ± 0.11	**32.11** ± 0.41	+9.3%	+7.0%
	Accuracy	28.46 ± 0.08	27.52 ± 0.19	27.69 ± 0.31	22.82 ± 0.45	29.83 ± 0.22	28.18 ± 0.46	29.26 ± 0.31	28.78 ± 0.14	**31.96** ± 0.35	+12.3%	+7.1%
Email	MicroF	41.25 ± 0.17	41.07 ± 0.33	35.83 ± 0.31	34.19 ± 0.13	27.81 ± 0.44	26.97 ± 0.39	41.97 ± 0.24	23.47 ± 0.25	**46.73** ± 0.37	+13.3%	+11.3%
	Accuracy	42.61 ± 0.31	42.20 ± 0.31	37.25 ± 0.16	35.13 ± 0.35	29.41 ± 0.46	27.16 ± 0.34	43.23 ± 0.30	25.84 ± 0.18	**47.70** ± 0.46	+11.9%	+10.3%

Table 3: Performance of link prediction w.r.t. classic base embedding models.

		Base	Biased walk	Additive	Additive-2	a la carte	a la carte-2	Nonce2vec	Dropout	meta-tail2vec	Improv. over Base	Improv. over 2nd best
DeepWalk as the base embedding model												
Wiki	MRR	75.28 ± 0.37	75.13 ± 0.41	75.81 ± 0.62	74.89 ± 0.78	76.31 ± 0.25	76.14 ± 0.33	67.42 ± 0.87	77.06 ± 0.71	**79.18** ± 0.52	+5.2%	+2.8%
	Hit@1	51.83 ± 0.42	52.04 ± 0.57	52.51 ± 0.67	51.48 ± 0.39	53.70 ± 0.61	53.59 ± 0.32	53.34 ± 0.49	54.19 ± 0.30	**57.22** ± 0.46	+10.4%	+5.6%
Flickr	MRR	50.05 ± 0.30	49.57 ± 0.19	49.80 ± 0.45	49.72 ± 0.41	50.36 ± 0.55	50.71 ± 0.65	50.83 ± 0.48	50.25 ± 0.59	**52.18** ± 0.61	+4.3%	+2.7%
	Hit@1	25.32 ± 0.24	25.63 ± 0.55	26.10 ± 0.41	26.55 ± 0.62	26.07 ± 0.30	26.39 ± 0.58	26.67 ± 0.33	26.19 ± 0.44	**28.11** ± 0.40	+11.0%	+5.4%
Email	MRR	44.17 ± 0.35	44.58 ± 0.26	44.52 ± 0.68	44.96 ± 0.28	44.49 ± 0.50	45.11 ± 0.34	44.80 ± 0.15	45.33 ± 0.08	**48.42** ± 0.55	+9.6%	+6.8%
	Hit@1	19.47 ± 0.38	19.96 ± 0.27	21.38 ± 0.15	21.66 ± 0.40	22.45 ± 0.58	22.63 ± 0.31	20.90 ± 0.44	23.02 ± 0.33	**24.31** ± 0.46	+24.9%	+5.6%
GraphSAGE as the base embedding model												
Wiki	MRR	81.36 ± 0.14	82.01 ± 0.10	80.56 ± 0.45	80.39 ± 0.21	81.82 ± 0.53	80.94 ± 0.62	82.18 ± 0.64	82.52 ± 0.40	**84.38** ± 0.61	+3.7%	+2.3%
	Hit@1	58.87 ± 0.52	58.39 ± 0.15	58.43 ± 0.61	58.92 ± 0.30	59.56 ± 0.29	59.34 ± 0.44	59.70 ± 0.37	59.93 ± 0.56	**62.04** ± 0.68	+5.4%	+3.5%
Flickr	MRR	55.83 ± 0.29	56.17 ± 0.36	55.04 ± 0.25	55.40 ± 0.58	56.28 ± 0.49	56.76 ± 0.40	56.31 ± 0.32	56.85 ± 0.71	**58.15** ± 0.43	+4.2%	+2.3%
	Hit@1	34.59 ± 0.52	35.15 ± 0.47	33.79 ± 0.38	33.36 ± 0.40	35.22 ± 0.68	35.29 ± 0.64	34.97 ± 0.50	35.74 ± 0.31	**36.92** ± 0.39	+6.7%	+3.3%
Email	MRR	46.71 ± 0.45	46.24 ± 0.29	46.05 ± 0.25	46.68 ± 0.44	47.03 ± 0.53	46.92 ± 0.30	47.18 ± 0.19	46.37 ± 0.60	**48.15** ± 0.44	+3.1%	+2.1%
	Hit@1	23.02 ± 0.23	22.73 ± 0.41	22.91 ± 0.44	22.65 ± 0.52	23.19 ± 0.39	23.14 ± 0.61	23.28 ± 0.43	23.07 ± 0.56	**24.55** ± 0.70	+6.6%	+5.4%

and reported with their standard deviations; best method appears in bold, and the second best approach is underlined.

5.2.1 Comparison of tail node embeddings. We first study classic base embedding models that are not specifically designed for robustness on sparse networks, followed by robust base models designed for sparse networks.

Classic base models. We report the performance of node classification in Table 2 w.r.t. classic base embedding models DeepWalk and GraphSAGE, respectively. On the one hand, our meta-tail2vec achieves significant improvements over the base embedding methods consistently, by 7.7%–10.9% w.r.t. DeepWalk and 9.3%–13.3% w.r.t. GraphSAGE in terms of MicroF. The results demonstrate that tail node embedding is a critical problem to address, and our proposed approach is indeed useful in refining the tail node embeddings. On the other hand, meta-tail2vec also outperforms other refinement baselines, gaining performance lifts in the range of 5.3%–11.3% over the best baseline in terms of microF. These baselines are sub-optimal as they only assume one model for all tail nodes, whereas meta-tail2vec can attribute its strong performance to the locality adaptation to each node under a meta-learning framework. In particular, the 2-hop variants of Additive and a la carte are often worse than their 1-hop models, which may be caused by noises from 2-hop nodes. However, our model also aggregates 2-hop nodes and attains better performance than its 1-hop version (as we will see in Sect. 5.3), potentially due to the local adaptation which can effectively filter noises at each locality.

We further report the performance of link prediction in Table 3 w.r.t. classic base embedding models. Similar conclusions can be drawn, where meta-tail2vec outperforms the base models by 3.1%–9.6% and the best baseline by 2.1%–6.8% in terms of MRR.

Robust base models for sparse networks. We also investigate whether meta-tail2vec can also improve base models designed for robustness on sparse networks.

We report the performance of node classification in Table 4, w.r.t. each of the base models SDNE, ARGA and DDGCN. While these base models are intended to handle sparse networks, they aim to increase the overall robustness of the learning process, and do not explicitly improve the embedding of the most vulnerable tail nodes. Thus, their performances on the tail nodes are not necessarily better

Table 4: Performance of node classification w.r.t. robust base embedding models (MiF for MicroF; Acc for accuracy).

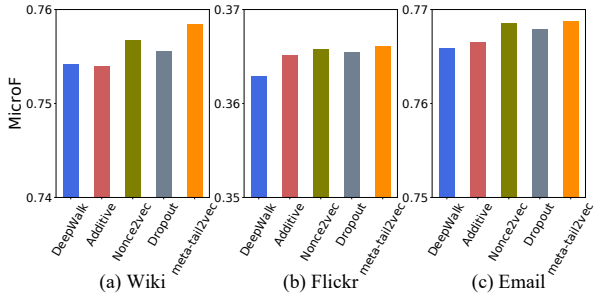
		Base	Additive	Nonce2vec	Dropout	meta-tail2vec
SDNE as the base embedding model						
Wiki	MiF	31.38 ± 0.34	34.46 ± 0.63	32.14 ± 0.75	34.69 ± 0.48	**37.99** ± 0.83
	Acc	34.10 ± 0.71	35.62 ± 0.28	34.59 ± 0.12	36.46 ± 0.43	**38.80** ± 0.64
Flickr	MiF	34.74 ± 0.86	35.49 ± 0.47	34.73 ± 0.37	35.38 ± 0.35	**38.50** ± 0.78
	Acc	32.67 ± 0.32	34.72 ± 0.28	33.59 ± 0.73	34.64 ± 0.49	**38.03** ± 0.66
Email	MiF	29.85 ± 0.48	31.07 ± 0.21	31.83 ± 0.46	34.50 ± 0.25	**47.21** ± 0.72
	Acc	32.90 ± 0.62	34.37 ± 0.27	33.79 ± 0.60	37.85 ± 0.47	**51.70** ± 0.33
ARGA as the base embedding model						
Wiki	MiF	32.22 ± 0.46	31.19 ± 0.23	32.47 ± 0.19	32.85 ± 0.23	**33.51** ± 0.31
	Acc	34.47 ± 0.52	33.84 ± 0.10	34.79 ± 0.21	35.22 ± 0.49	**35.80** ± 0.23
Flickr	MiF	24.60 ± 0.15	23.69 ± 0.18	25.16 ± 0.20	24.71 ± 0.15	**25.93** ± 0.25
	Acc	22.81 ± 0.17	21.59 ± 0.11	24.26 ± 0.46	23.65 ± 0.39	**25.37** ± 0.16
Email	MiF	24.38 ± 0.31	23.94 ± 0.47	24.97 ± 0.35	25.48 ± 0.53	**26.11** ± 0.25
	Acc	26.57 ± 0.43	25.69 ± 0.30	27.02 ± 0.37	27.54 ± 0.21	**27.95** ± 0.16
DDGCN as the base embedding model						
Wiki	MiF	29.49 ± 0.18	27.68 ± 0.58	31.20 ± 0.44	30.37 ± 0.35	**33.02** ± 0.43
	Acc	31.39 ± 0.25	30.82 ± 0.21	33.87 ± 0.75	32.69 ± 0.40	**36.27** ± 0.41
Flickr	MiF	28.57 ± 0.47	26.92 ± 0.08	30.09 ± 0.35	29.17 ± 0.26	**31.03** ± 0.52
	Acc	25.90 ± 0.71	24.44 ± 0.12	26.76 ± 0.49	26.37 ± 0.25	**28.38** ± 0.54
Email	MiF	38.95 ± 0.67	38.73 ± 0.55	39.62 ± 0.43	39.15 ± 0.40	**41.83** ± 0.34
	Acc	39.81 ± 0.56	38.20 ± 0.35	42.32 ± 0.63	41.69 ± 0.41	**44.13** ± 0.73

Table 5: Performance of link prediction w.r.t. robust base embedding models (H@1 for hit@1).

		Base	Additive	Nonce2vec	Dropout	meta-tail2vec
SDNE as the base embedding model						
Wiki	MRR	72.25 ± 0.48	72.53 ± 0.30	74.44 ± 0.39	75.08 ± 0.65	**76.97** ± 0.61
	H@1	52.19 ± 0.27	51.94 ± 0.39	54.50 ± 0.61	55.21 ± 0.35	**57.58** ± 0.74
Flickr	MRR	46.82 ± 0.20	47.09 ± 0.44	48.35 ± 0.51	48.17 ± 0.29	**49.31** ± 0.46
	H@1	26.23 ± 0.16	27.00 ± 0.33	28.82 ± 0.61	28.39 ± 0.10	**29.26** ± 0.20
Email	MRR	34.02 ± 0.76	34.29 ± 0.51	36.87 ± 0.49	36.42 ± 0.55	**39.55** ± 0.50
	H@1	17.51 ± 0.24	18.65 ± 0.51	21.19 ± 0.40	20.88 ± 0.13	**22.86** ± 0.63
ARGA as the base embedding model						
Wiki	MRR	48.57 ± 0.40	46.49 ± 0.38	49.16 ± 0.45	50.27 ± 0.14	**51.08** ± 0.20
	H@1	41.40 ± 0.52	40.49 ± 0.07	41.67 ± 0.35	42.22 ± 0.10	**43.73** ± 0.65
Flickr	MRR	35.52 ± 0.32	35.41 ± 0.72	35.69 ± 0.63	36.31 ± 0.28	**36.87** ± 0.45
	H@1	29.73 ± 0.34	28.86 ± 0.40	29.89 ± 0.62	30.51 ± 0.77	**31.37** ± 0.29
Email	MRR	26.83 ± 0.29	25.89 ± 0.47	26.91 ± 0.18	26.22 ± 0.40	**27.26** ± 0.55
	H@1	16.51 ± 0.29	16.30 ± 0.42	17.22 ± 0.40	16.89 ± 0.31	**17.87** ± 0.35
DDGCN as the base embedding model						
Wiki	MRR	73.25 ± 0.49	74.10 ± 0.34	74.28 ± 0.15	74.92 ± 0.53	**75.31** ± 0.67
	H@1	51.28 ± 0.39	50.77 ± 0.21	51.86 ± 0.45	52.56 ± 0.32	**53.30** ± 0.61
Flickr	MRR	52.17 ± 0.40	50.74 ± 0.51	52.23 ± 0.42	51.79 ± 0.60	**52.49** ± 0.34
	H@1	37.15 ± 0.38	35.82 ± 0.85	37.53 ± 0.42	37.16 ± 0.60	**38.68** ± 0.63
Email	MRR	41.58 ± 0.45	40.83 ± 0.37	42.96 ± 0.39	42.81 ± 0.12	**43.47** ± 0.18
	H@1	27.35 ± 0.39	28.31 ± 0.63	28.58 ± 0.25	28.87 ± 0.30	**29.22** ± 0.36

than classic base models. Note that our approach meta-tail2vec is embedding-agnostic, meaning that even for base models already designed for sparse networks, we can still refine their tail node embeddings, as demonstrated by the results that meta-tail2vec outperforms the base embeddings by an average of 14.9% in terms of MicroF on node classification. On the other hand, we also compare meta-tail2vec to other baseline refinement methods. (Due to space constraint, we only present the results of Addictive, Nonce2vec and Dropout, which are generally the best baselines among all.) Again, due to our locality-aware task formulation, the meta-learning strategy is able to adapt to the locality of each tail node well, resulting in an average performance lift of 8.5% in terms of MicroF compared to the best baseline.

Furthermore, we report the performance of link prediction in Table 5. We observe similar performance comparisons, where on average meta-tail2vec outperforms the robust base models by 5.2% and the best baseline by 2.0% in terms of MRR.

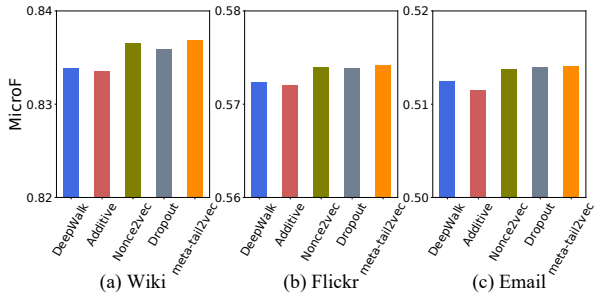


(a) Wiki (b) Flickr (c) Email

Figure 4: Performance of node classification on head nodes w.r.t. DeepWalk as the base embedding model.

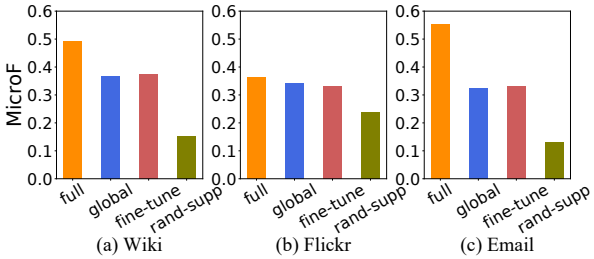
5.2.2 Comparison of head node embeddings. While our main goal is to improve tail node embeddings, we further evaluate the performance on the head nodes to validate that their embeddings still remain competitive. In theory head node embeddings are not changed in any way, since we only predict new embedding vectors for tail nodes. However, the performance of head nodes on a downstream application can be potentially improved, as training the downstream model can still benefit from the improved quality of tail nodes.

Thus, we further conducted an experiment on head node embeddings. We sampled and evaluated a test set comprising 10% of the head nodes, and trained a model for each downstream task on the other nodes, inclusive of the tail nodes and the remaining 90% head nodes. We tabulate the results in Figs. 4 and 5 for node classification and link prediction, respectively. As hypothesized, both meta-tail2vec and other baseline refinement approaches can slightly outperform the base embedding model, due to the improved quality of tail node embeddings in the downstream training data. However,



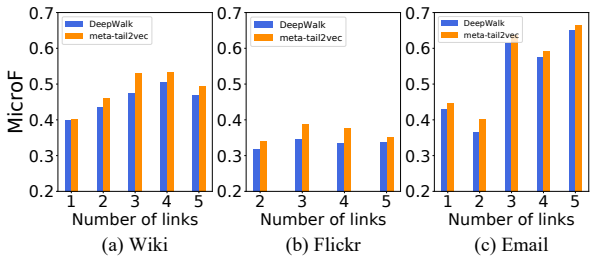
(a) Wiki (b) Flickr (c) Email

Figure 5: Performance of link prediction on head nodes w.r.t. DeepWalk as the base embedding model.



(a) Wiki (b) Flickr (c) Email

Figure 6: Ablation study of the meta-learning strategy on node classification w.r.t. DeepWalk as the base model.



(a) Wiki (b) Flickr (c) Email

Figure 8: Impact of number of links on node classification w.r.t. DeepWalk as the base model.

it is not surprising that the improvements are modest compared to tail nodes, in the range of 0.3%–0.9% only. The reason is that the head node embeddings remain unchanged and their performance is only indirectly influenced by refined tail node embeddings. Nevertheless, we validated the goal of significantly improving tail node embeddings whilst head node embeddings remain robust.

5.3 Model analysis and discussion

5.3.1 Ablation study. We analyze the contribution of our approach by an ablation study. Using DeepWalk as the base embedding model, we compare the following four variants of meta-tail2vec: (1) *full*, the full meta-tail2vec model; (2) *global*, only train one global embedding regression model on the head nodes, and predict the embedding vectors of all tail nodes with the same global model (equivalent to the Dropout baseline); (3) *fine-tune*, fine-tune the pre-trained global model on the support set of a tail node before predicting its embedding vector; (4) *rand-supp*, the same approach as the full model except the locality awareness, which samples random nodes from the graph as the support sets.

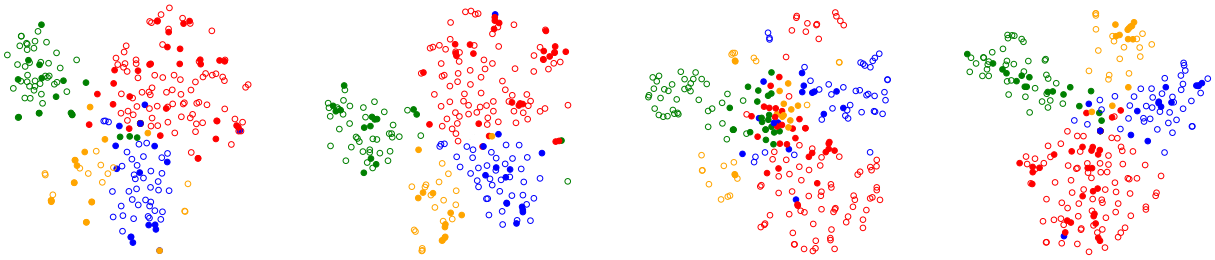
Their performances are reported in Fig. 6. Among the four variants, we observe that the fine-tune model is only able to achieve marginally better performance than the global model, as fine-tuning on the small support set of a tail node can easily cause overfitting. In particular, on the Flickr dataset, the fine-tune model is in fact slightly worse than the global model due to overfitting. Next, the rand-supp model performs the worst, implying that the locality-aware task generation is critical for graph data. Finally, the full model performs the best, demonstrating the effectiveness of our locality-aware meta-learning approach.

5.3.2 Visualization. We study how exactly meta-tail2vec updates the base embeddings for the tail nodes. In particular, we visualize the base and refined embeddings using the t-SNE algorithm.

We first showcase two base embedding models DeepWalk and SDNE in Figs. 7(a) and 7(c), respectively. While both base models can separate the head nodes (denoted by hollow dots) into relatively well defined clusters corresponding to their ground truth classes (as coded by different colors), they fail at the tail nodes (denoted by solid dots). There is a significant mixture of tail nodes from different classes at the center of the visualization for both DeepWalk and SDNE, implying that they are not designed to work well on tail nodes. On the other hand, we examine the refined embeddings by meta-tail2vec w.r.t. the two base models in Figs. 7(b) and 7(d) respectively. As head nodes already have high-quality base embeddings, the key is to improve the tail node embeddings. Our approach specifically refines the tail nodes so that they now move towards the cluster center of their class together with the head nodes from the same class. The contrast with the base embeddings conclude that our proposed approach is effective in learning tail node embeddings.

5.3.3 Impact of tail node sparsity. We breakdown the performance of tail nodes by their number of links. As shown in Fig. 8, the performance improvement is observed across the spectrum for tail nodes with between 1 and 5 links. Note that on the Flickr dataset, all nodes have at least 2 links. The improvement is generally smaller on nodes with only one link as expected, given that meta-tail2vec is also constrained by the very limited structural information.

5.3.4 Impact of neighborhood hops. Next, we analyze the impact of number of hops used for computing the input feature vector in



(a) DeepWalk (b) meta-tail2vec on DeepWalk (c) SDNE (d) meta-tail2vec on SDNE

Figure 7: Visualization of base embeddings by DeepWalk and SDNE, and their respective refinement by meta-tail2vec on the Email dataset. *Solid* points denote tail nodes and *hollow* points denote head nodes. Each color represents one class.

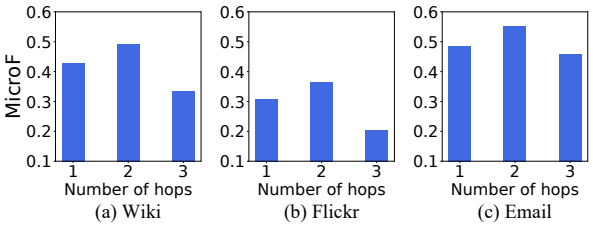


Figure 9: Impact of number of hops on node classification

Eq. (5). The results across different number of hops are presented in Fig. 9. In particular, aggregating from the 2-hop neighborhood achieves optimal performance on all datasets. In particular, aggregating from the 3-hop neighborhood results in decreased performance due to more noises from the less relevant nodes.

6 CONCLUSION

In this paper, we investigated the problem of tail node embedding on graphs. We first formulated the problem as an instance of few-shot regression, and proposed a novel approach *meta-tail2vec* for refining tail node embeddings. In particular, to personalize each tail node given its local contexts, we designed a locality-aware task generation strategy to capture the prior knowledge across nodes at different localities. Finally, extensive experiments demonstrated the promising performance of meta-tail2vec on both node classification and link prediction.

ACKNOWLEDGEMENT

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG-RP-2018-001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

REFERENCES

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *TACL* 5 (2017), 135–146.
[2] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *TKDE* 30, 9 (2018), 1616–1637.
[3] Ruichu Cai, Xuexin Chen, Yuan Fang, Min Wu, and Yuexing Hao. 2020. Dual-Dropout Graph Convolutional Network for Predicting Synthetic Lethality in Human Cancers. *Bioinformatics* (2020).
[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*. 891–900.
[5] Jatin Chauhan, Deepak Nathani, and Manohar Kaul. 2020. Few-Shot Learning on Graphs via Super-Classes based on Graph Spectral Measures. In *ICLR*.
[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*. 4171–4186.
[7] Yuan Fang, Wenqing Lin, Vincent W Zheng, Min Wu, Kevin Chen-Chuan Chang, and Xiao-Li Li. 2016. Semantic proximity search on graphs with metagraph-based learning. In *ICDE*. 277–288.
[8] Yuan Fang, Wenqing Lin, Vincent W Zheng, Min Wu, Jiaqi Shi, Kevin Chang, and Xiaoli Li. 2019. Metagraph-based Learning on Heterogeneous Graphs. *TKDE* (2019).
[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*. 1126–1135.
[10] Victor Garcia and Joan Bruna. 2018. Few-shot learning with graph neural networks. In *ICLR*.
[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. 855–864.
[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.
[13] Xu Han, Hao Zhu, Pengfei Yu, Ziyun Wang, Yuan Yao, Zhiyuan Liu, and Maosong Sun. 2018. FewRel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. In *EMNLP*. 4803–4809.
[14] Aurélie Herbelot and Marco Baroni. 2017. High-risk learning: acquiring new word vectors from tiny data. In *EMNLP*. 304–309.
[15] Ziniu Hu, Ting Chen, Kai-Wei Chang, and Yizhou Sun. 2019. Few-Shot Representation Learning for Out-Of-Vocabulary Words. In *ACL*. 4102–4112.
[16] Mikhail Khodak, Nikunj Saunshi, Yingyu Liang, Tengyu Ma, Brandon Stewart, and Sanjeev Arora. 2018. A La Carte Embedding: Cheap but Effective Induction of Semantic Feature Vectors. In *ACL*. 12–22.
[17] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[18] Angeliki Lazaridou, Marco Marelli, and Marco Baroni. 2017. Multimodal word meaning induction from minimal exposure to natural text. *Cognitive science* 41 (2017), 677–705.
[19] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. MeLU: Meta-Learned User Preference Estimator for Cold-Start Recommendation. In *SIGKDD*. 1073–1082.
[20] Jingjing Li, Mengmeng Jing, Ke Lu, Lei Zhu, Yang Yang, and Zi Huang. 2019. From zero-shot learning to cold-start recommendation. In *AAAI*. 4189–4196.
[21] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Learning to propagate for graph meta-learning. In *NeurIPS*. 1037–1048.
[22] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. 2019. Learning to Propagate Labels: Transductive Propagation Network for Few-shot Learning. In *ICLR*.
[23] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *SIGKDD*. 1105–1114.
[24] 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*. 695–704.
[25] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2018. Adversarially Regularized Graph Autoencoder for Graph Embedding. In *IJCAI*. 2609–2615.
[26] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. 701–710.
[27] Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. 2017. Mimicking word embeddings using subword RNNs. In *EMNLP*. 102–112.
[28] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *ICML*. 1842–1850.
[29] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *NeurIPS*. 4077–4087.
[30] Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *ACL*. 1556–1566.
[31] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.
[32] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *NeurIPS*. 3630–3638.
[33] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. 2017. DropoutNet: Addressing cold start in recommender systems. In *NeurIPS*. 4957–4966.
[34] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *SIGKDD*. 1225–1234.
[35] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. GraphGAN: Graph representation learning with generative adversarial nets. In *AAAI*. 2508–2515.
[36] Jian Wei, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. 2017. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications* 69 (2017), 29–39.
[37] Wenhan Xiong, Mo Yu, Shiyu Chang, Xiaoxiao Guo, and William Yang Wang. 2018. One-shot relational learning for knowledge graphs. In *EMNLP*. 1980–1990.
[38] Carl Yang, Mengxiong Liu, Vincent W Zheng, and Jiawei Han. 2018. Node, motif and subgraph: Leveraging network functional blocks through structural convolution. In *ASONAM*. 47–52.
[39] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *IJCAI*. 2111–2117.
[40] 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*. 6656–6663.
[41] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. 2017. Local Higher-Order Graph Clustering. In *SIGKDD*. 555–564.
[42] Chuxu Zhang, Huaxiu Yao, Chao Huang, Meng Jiang, Zhenhui Li, and Nitesh V Chawla. 2020. Few-Shot Knowledge Graph Completion. In *AAAI*. 3041–3048.
[43] 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*. 2357–2360.